VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

WEBOVÉ ROZHRANÍ PRO SLEDOVÁNÍ PROVOZU V BEZDRÁTOVÝCH SÍTÍCH

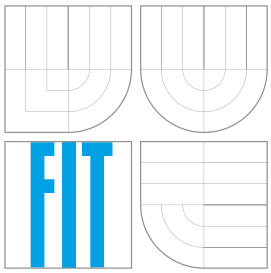
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN GÁBOR

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

WEBOVÉ ROZHRANÍ PRO SLEDOVÁNÍ PROVOZU V BEZDRÁTOVÝCH SÍTÍCH

WEB INTERFACE FOR WIRELESS NETWORK MONITORING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MARTIN GÁBOR
AUTHOR

VEDOUCÍ PRÁCE — Ing. FRANTIŠEK ZBOŘIL, Ph.D.
SUPERVISOR

BRNO 2010

Abstrakt

Cílem této diplomové práce je analyzovat, navrhnout a vytvořit architekturu pro webové rozhraní systému WSageNt. Jeho hlavní činností bude monitorování provozu a spravování topologie sítě systému. Práce popisuje základní technologie, principy návrhu a způsoby implementace.

Abstract

The aim of this diploma thesis is to analyze, design and create the architecture of the WSageNt system web interface. The main focus of the system will be traffic monitoring and topology control of the network. The work describes basic technologies, design principles and implementation methods.

Klíčová slova

bezdrátová síť, monitorování, provoz, komunikace, topologie, senzor, RSSI, TinyOs, micaz, iris, WSageNt

Keywords

wireless network, monitoring, traffic, communication, topology, senzor, RSSI, TinyOS, micaz, iris, WSageNt

Citace

Martin Gábor: Webové rozhraní pro sledování provozu v bezdrátových sítích, diplomová práce, Brno, FIT VUT v Brně, 2010

Webové rozhraní pro sledování provozu v bezdrátových sítích

Prohlášení

Prehlasujem, že som túto diplomvú prácu vypracoval samostatne pod vedením p. Ing. Františka Zbořila, Ph.D. Ďalšie informácie mi poskytol Ing. Jan Horáček. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Gábor
26.5.2010

Poděkování

Týmto by som chcel poďakovať hlavne môjmu vedúcemu práce, pánovi Ing. Zbořilovi, Ph.D., za odbornú pomoc a poskytnuté konzultácie. Moje poďakovanie patrí aj pánovi Ing. Janu Horáčkovi za uvedenie do problematiky a jeho odbornú pomoc.

Obsah

Kapitola 1

Úvod

Už od nepamäti ľudia používali rôzne formy komunikácie. Okrem ústnych či písomných foriem, jestvovalo i dorozumievanie pomocou rôznych symbolov a znamení. Napríklad mávnutím bielej zástavy sa bojovník vzdal na bojisku svojmu nepriateľovi, či rôzne dymové signály, alebo zrod posunkovej reči pre hluchonemých. Ak nefungovalo rádiové spojenie, lode sa jednoducho dorozumeli Morseovou abecedou vyslanou svetelnými reflektormi. Existuje mnoho ďalších príkladov z minulosti, ktoré bezpochyby ovplyvnili vznik bezdrôtových technológií.

Paradoxne, zavedenie drôtových sietí a ich použitia ako prenosového média pre komunikáciu, naštartovalo vytvorenie informačnej infraštruktúry v 20. storočí. Bol to zásadný pokrok v technológiách, ktorý umožnil mnoho zmien vo viacerých sférach života. [11]

Postupne sa drôtové siete začali nahradzovať bezdrôtovými, najmä v oblastiach telekomunikačných technológií, čo umožnilo mobilitu a portabilitu. V 80. rokoch 20. storočia sa začali používať bezdrôtové telefóny. Spočiatku to boli analógové prenosy, neskôr digitálne. Následne sa do bežného života zaviedli mobilné siete, ktorým výrazne dopomohla sieť GSM. Rozšírila sa do viac ako 184 krajín s viac ako 747 miliónmi užívateľov. Bezdrôtové systémy boli využité aj v satelitných prenosoch a LAN, MAN a PAN sieťach atď. [19]

Špecifickým prípadom sietí sú senzorové – WSN, ktoré využívajú snímacie receptory na zachytávanie zmien vo svojom okolí. Je možné ich implementovať ako drôtovú sieť s určitou štruktúrou. V praxi sa však viac uplatnili bezdrôtové senzorové siete, skladajúce sa zo zariadení, na ktorých je spustený nejaký prevádzkový program. Dôležitým parametrom je spotreba zariadenia, s ktorým súvisí typ napájania. Okrem toho, hrá dôležitú úlohu aj veľkosť, aby bolo možné integrovať zariadenie do rôznych vstavaných a iných menších systémov. V závislosti od týchto parametrov môžeme nájsť mnoho uplatnení v bežnom živote pre bezdrôtové senzorové siete.

Aj WSageNt je jedným z takých projektov, ktoré aplikujú technológie WSN sietí. Ide o snahu vytvoriť jednoduchý systém, ktorý by poskytoval kompletnú obsluhu senzorových zariadení firmy Crossbow a hľadal by možnosti jeho integrovania do každodenného života.

Okrem samotného využívania vlastností akejkoľvek siete by sme mali vždy mať prehľad o jej stave a zmenách, ktoré sa v nej dejú. Je to dôležité napr. na odhaľovanie porúch, ktoré sa môžu vyskytnúť najmä ak sú uzly siete autonómne. V bezdrôtových sieťach sa okrem toho môže meniť jej topológia, čo môže spôsobiť neočakávané správanie. Obmedzenia môžu byť spôsobené aj zahltením niektorých uzlov, preto je vhodné monitorovať aj komunikáciu. Z hľadiska energie môžu byť senzorové zariadenia obzvlášť na to náchylné, preto by sme sa mali snažiť čo najefektívnejšie spravovať takýto systém a využiť výhody, ktoré nám ponúka.

Webové technológie sú čoraz viac integrované do rôznych komplexných systémov, kde

nie sú kladené vysoké požiadavky na grafickú dynamiku. Pomocou nich dokážeme rýchlo a efektívne vytvoriť prezentačnú vrstvu aplikácie nezávislú na platforme. V súčasnosti je veľmi rozšírená podpora pre dynamické generovanie obsahu tejto vrstvy spojená so silnou aplikačnou logikou. Spolu umožňujú vytvárať mocné a bezpečné aplikácie na architektúre klient–server.

V tejto práci bude vysvetlené, ako je možné využiť webové technológie na správu bezdrôtovej senzorovej siete. V úvode sa oboznámime so základnými princípmi a analyzujeme prostriedky, ktoré budeme používať. Ďalej navrhneme architektúru systému, ktorá bude podporovať všetky požadované funkcie. Nakoniec vysvetlíme princípy implementácie a zhrnieme výsledok práce.

Kapitola 2

Bezdrôtové systémy a siete

Aj keď história bezdrôtových sietí presahuje tisíce rokov, bezdrôtový prenos sa veľmi rozšíril za posledných 15 až 25 rokov. Je to najrýchlejšie sa rozrastajúci segment telekomunikačného priemyslu. Siete tohto typu sú používané v každodennom osobnom a profesionálnom živote ľudí. Očakáva sa, že počet užívateľov bezdrôtových systémov prevýši počet užívateľov s káblovou infraštruktúrou. Je to spôsobené najmä výhodami, ktoré tieto technológie prinášajú. Je to najmä mobilita a cena. [16]

Táto kapitola sa ponesie v duchu základov týchto systémov a uvedie základnú problematiku. Keďže je to rozsiahla téma, viac sa budeme venovať najmä tým vlastnostiam technológie, ktoré sú podstatné pre vyvíjaný projekt.

Úvodom treba načrtnúť niečo z histórie týchto zariadení. Neskôr sa bude hovoriť detailnejšie o dôležitých zložkách týchto bezdrôtových systémov.

2.1 Generácie bezdrôtových systémov

Od roku 1970 až po súčasnosť môžeme, podľa technológii prístupu k prenosovému kanálu, bezdrôtové systémy klasifikovať nasledovne [2]:

- 1G bezdrôtové systémy, založené na protokole FDMA.
- 2G bezdrôtové systémy, založené na TDMA a CDMA.
- 3G bezdrôtové systémy, najčastejšie založené na W–CDMA.
- 4G bezdrôtové systémy, založené na Multi–Carrier – CDMA alebo OFDM (TDMA).

Väčšinou sa toto rozdelenie spája s mobilnými bezdrôtovými systémami, ktoré však dopomohli k technologickému vývoju bezdrôtových sietí všeobecne. Spočiatku to boli analógové siete, založené na bunkách, určené výhradne na prenos hlasu. Kvalita však nebola príliš dobrá, preto ju nahradili digitálne systémy už v druhej fáze. Tie poskytovali väčšiu prenosovú kvalitu, systémovú kapacitu a aj dosah. Najznámejším systémom založeným na TDMA bol a stále je **GSM**. Okrem odlišných prístupov k prenosovému médiu sa v ďalších fázach vyvinulo aj niekoľko protokolov, ktoré zlepšovali vlastnosti bezdrôtových sietí. Ich prínosom bolo vysoká rýchlosť a kvalita bezdrôtových prenosov [2].

Podrobnejšie sa budeme protokolom prístupu k prenosovému kanálu venovať v kapitole 2.3.2.

2.2 Obmedzenia bezdrôtových technológií

Nevýhodami bezdrôtových technológií sú nasledujúce výkonnostné obmedzenia [2]:

- Vysielacie pásmo (dosah signálu).
- Energia použitá na vytvorenie signálu.
- Mobilita.
- Šírka pásma.
- Skutočný bitový tok.

Vysielacie pásmo je kritickým faktorom. Je to oblasť pokrytia signálom od vysielača k prijímaču. To je úzko spojené so silou signálu. Čím viac sa bude zväčšovať vzdialenosť medzi vysielačom a prijímačom, tým slabší bude signál. S tým môže stúpať chybovosť prenášaných dát.

Mobilita prijímačov závisí od ich samotnej veľkosti. Menšie zariadenia poskytujú lepšiu pohyblivosť. To môže byť dosiahnuté zmenšením batérií a tým aj spotreby energie, avšak na úkor sily vysielacieho signálu.

Šírka pásma predstavuje množstvo frekvenčného spektra dostupného pre jedného užívateľa. Pri použití širších kanálov sa zväčší. Časť spektra môžeme využiť na opravu chýb *(Error Correction)*.

Skutočný bitový tok najviac závisí na šírke pásma dostupného pre jedného užívateľa. Okrem toho tiež závisí na niekoľkých iných faktoroch, ktoré ho ovplyvňujú, ako napr. pohyb vysielača alebo hustota prijímačov v určitej lokalite. Skutočný bitový tok je zvyčajne väčší u stacionárnych prijímačov ako u pohybujúcich sa. Dôvodom je práve časť spektra, ktorú používame na opravu chýb. Musí byť väčšia v závislosti od rýchlosti pohybu, pretože pri signáloch pohybujúcich sa objektov vzniká väčšia **interferencia** [1].

Pozícia objektu tiež ovplyvňuje interferenciu vlnenia. Optimálnou polohou prijímača je, ak je priamočiaro viditeľný s vysielačom, pričom prijímač sa nenachádza ďaleko od vysielača [2]. Žiaľ, vždy takéto podmienky nie sú, preto sa musíme vysporiadať s problémami, ktoré táto technológia prináša.

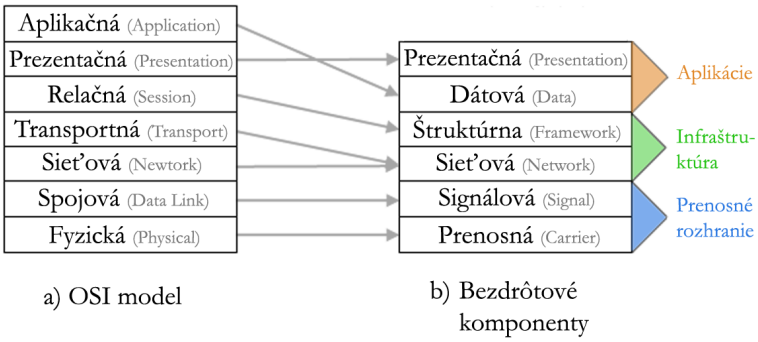
2.3 Vrstvy siete

Medzinárodná organizácia pre štandardizáciu (*ISO*) vytvorila zaužívaný model sieťovej architektúry, nazývaný **OSI referenčný model**. V ňom definuje sedem vrstvových modulov sieťového systému. Každá vrstva poskytuje určitú správu. Spolu s ostatnými definuje úplnú sieťovú konektivitu. V bezdrôtových systémoch je tento model aplikovateľný a pri jeho použití by zabezpečoval úplnú funkcionalitu. Obrázok 2.1 vyjadruje aplikovanie tohto modelu.

OSI model však nebol navrhnutý pre bezdrôtové a mobilné systémy, aj keď pre mnohé systémy môže byť do určitého stupňa použitý. Táto štruktúra môže byť nepostačujúca najmä pre novšie technológie [11].

V nasledujúcich podkapitolách budú uvedené základy iba tých nižších vrstiev modelu pre bezdrôtové siete, ktoré sa výrazne odlišujú od základov v drôtových sieťach.

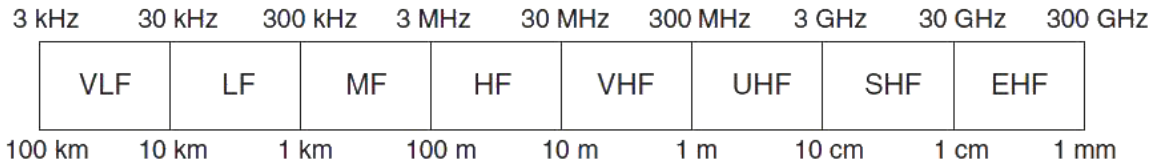
[1] vzájomné ovplyvňovanie viacerých vĺn



Obrázek 2.1: OSI model a jeho úprava pre bezdrôtové systémy [11]

2.3.1 Fyzická vrstva

Základným rozdielom drôtovej a bezdrôtovej siete je médium, ktorým sa šíria dáta. Tak ako v prípade drôtových sietí, v ktorých má užívateľ možnosť vybrať konkrétnu technológiu, tak aj v bezdrôtových je tomu tak. Pri návrhu je dôležité vybrať tú správnu, ktorá definuje **prenosovú frekvenciu**. Každá prenosová frekvencia obsahuje určité **propagačné charakteristiky**. Pre komunikačné účely to nebýva len jedna frekvencia, ale určité **frekvenčné pásmo**. Pri rádiových prenosoch je toto pásmo rozdelené od 3kHz až po 300Ghz na osem častí, tak ako je znázornené na obrázku 2.2 Možnosťou sú aj **infračervené** (od 300GHz – 120THz) a **optické** (120THz – 385THz) komunikácie. Rozdelený frekvenčných rozsah je predmetom regulácie, aby sa zabránilo nežiadaným rušeniam medzi rozdielnymi užívateľmi a systémami.



Obrázek 2.2: Elektromagnetické spektrum, rádiové frekvencie [12]

Dôležitým prvkom v bezdrôtovej komunikácií je **modulácia a demodulácia**. Keď si dve zariadenia vymieňajú dáta, prenášajú sa sekvencie symbolov. Každý symbol patrí do určitej kanálovej abecedy. Pri procese modulácie, symbol alebo skupina symbolov z kanálovej abecedy je zobrazená do jednej z konečného počtu vĺn s rovnakou vzdialenosťou – dobou trvania symbolu. Existujú tri elementárne typy modulácie digitálneho signálu. Sú to **ASK** (*Amplitude Shift Keying*), **PSK** (*Phase Shift Keying*) a **FSK** (*Frequency Shift Keying*), ktoré môžu byť použité aj v rôznych kombináciách. Na opačnej strane sa nachádza prijímač, ktorý musí prichádzajúci signál demodulovať. Z dôvodu vedľajších efektov propagácie signálu je prijatý signál skreslený, preto nemôže prijímač jednoznačne určiť prijatý symbol. Vyhodnotenie prebieha na základe **chybovosti symbolu** (*Symbol Error Rate*).

Chyby, ktoré vznikajú pri komunikácií, sú spôsobené vplyvom mnohých **fyzikálnych fenoménov**, ktoré modulujú prenášaný signál. Medzi ne patrí hlavne **šum**, **interferencia**, **odraz**, **rozptýlenie**, **Dopplerov jav** a **oslabenie signálu**.

Rádiové antény vyžarujú signál do všetkých smerov (takmer) rovnako veľkou silou. Tomuto typu hovoríme **všesmerové**. Druhým typom sú **orientované** antény. V obidvoch

prípadoch neprichádza do prijímača len jedna, ale množstvo kópií toho istého signálu z rôznych strán a trás, ktoré sú rôzne dlhé, s rôznou silou signálu. Tomuto javu sa hovorí **MultiPath propagácia**, kedy existuje priamy signál (**LOS** - *Line Of Sight*) a odrazený (**NLOS** - *Non line Of Sight*). Tento problém znázorňuje obrázok 2.3. [12]

Obrázek 2.3: MultiPath Fading problém [12]

2.3.2 Spojová vrstva

Spojová vrstva sa často nazýva vrstva **MAC** (*Medium Access Control*), kvôli prítomnosti týchto protokolov. Rieši zdanlivo jednoduché úlohy prístupu k médiu. Medzi najdôležitejšie požiadavky patrí **priepustnosť**, **stabilita**, **efektivita**, **spravodlivá dostupnosť**, **krátka doba prístupu** (čas medzi príchodom paketu a prvým pokusom doručiť ho), **krátka doba prenosu** (čas medzi príchodom paketu a jeho úspešným doručením) a **nízka réžia**. V prípade obsluhy každého paketu alebo prídavných kontrolných paketov, môže réžia v MAC protokoloch spôsobiť **kolízie**. Kolízie môžu vzniknúť aj prípade, ak protokol umožňuje posielanie paketov dvom alebo viacerým uzlom v tom istom čase. Tieto kolízie môžu spôsobiť, že prijímač nesprávne dekóduje pakety.

Operácie a výkon MAC protokolov sú veľmi závislé a ovplyvňované vlastnosťami nižšej – fyzickej vrstvy. Sú spoluúčastníkmi na riešení problémov vznikajúcich pri bezdrôtovej komunikácií. Jedným problémom je **časový faktor** a niekedy aj celkom vysoká **chybovosť**, vyplývajúca z fyzikálnych a prírodných fenoménov (viď kapitola 2.3.1). [12]

Protokoly pevného priradenia prístupu

V tejto triede protokolov sú dostupné zdroje prideľované medzi uzly na určitý časový interval tak, že ich môžu využívať výhradne iba oni bez rizika kolízie. Doba môže byť určená na jednotky minút, hodín alebo dokonca dlhšie, alebo naopak na desiatky milisekúnd a pod. Typickými predstaviteľmi tejto triedy sú protokoly **TDMA** (*Time division multiple access*), **FDMA** (*Frequency division multiple access*), **CDMA** (*Code division multiple access*) a **SDMA** (*Space division multiple access*). [12]

Protokoly prístupu na požiadanie

V tejto triede protokolov musia uzly požiadať o pridelenie prostriedkov, ktoré je exkluzívne a to väčšinou na krátku dobu získania dát a iných operácií. V prípade úspešnej alokácie je potvrdenie zaslané žiadajúcemu uzlu spolu s informáciami o alokovaných zdrojoch. Napríklad počet časových slotov v TDMA systéme a doby ich priradenia.

Protokoly tejto triedy môžu byť centralizované alebo distribuované. Príklady centralizovaných protokolov sú **HIPERLAN/2**, **DQRUMA** alebo **MASCARA**. Uzol pošle požiadavku centrálnemu uzlu, ktorý ju buď potvrdí, alebo zamietne. Príkladom distribuovaných sú protokoly chodu tokenov, ako napr. **IEEE 802.4 Token Bus**, v ktorých je sieť organizovaná do kruhu. V ňom si uzly posielajú tokeny. [12]

Protokoly náhodného prístupu

Uzly sú nekoordinované, takže protokoly sú úplne distribuované. Protokoly náhodného prístupu (*Random Access Protocols*) často zahŕňajú náhodnú veličinu, napr. pri nastavovaní časovačov a pod. Jeden z prvých a veľmi dôležitých je protokol **ALOHA** alebo jeho druhá verzia **slotted ALOHA**, vyvinutý na Univerzite v Havaji. V pôvodnom Aloha protokole platí, že každý uzol, ktorý chce odoslať paket, tak urobí okamžite. Neexistuje žiadna koordinácia s ostatnými uzlami, takže protokol akceptuje riziko kolízie na strane prijímača. Ten musí odpovedať okamžite, že paket dorazil úspešne. Ak táto odpoveď nepríde na opačnú stranu, odosielateľ pochopí, že nastala kolízia a v náhodnom čase sa pokúsi odoslať paket znovu.

Ďalším príkladom je protokol **CSMA** (*Carrier Sense Multiple Access*). K zabráneniu kolízie sa používa počúvanie a rešpektovanie prijímacieho uzlu. Ten môže byť dostupný alebo nedostupný. Na základe toho môžu ostatné uzly s ním komunikovať. Môže sa stať, že sa dva uzly pokúsia vyslať paket v takmer tom istom čase, pričom žiadny nedetekuje nedostupnosť, dôsledkom čoho vznikne kolízia. Na jej zabránenie sa používajú rôzne varianty tohto protokolu - **CSMA/CA** (*Carrier Sense Multiple Access With Collision Avoidance*), **CSMA/CD** (*Carrier Sense Multiple Access With Collision Detection*) a pod.

Ďalším typom je **RTS/CTS handshake**. Tento protokol je založený na protokole **MACAW**. Používa jednoduchý kanál a dva špeciálne kontrolné pakety. Predpokladajme, že uzol B chce odoslať dáta uzlu C. Hneď ako získa prístup ku kanálu, odošle paket s požiadavkou na zaslanie – **RTS** (*Request To Send*), ktorá zahŕňa dobu trvania celkovej transakcie. Ak C dostal tento paket v poriadku, odpovie paketom o možnosti zaslania – **CTS** (*Clear To Send*). Ak B získa tento paket, inicializuje komunikáciu. Na záver C odpovie paketom o potvrdení prijatých dát. [12]

2.4 Signál a jeho sila v prostredí

Sila získaného signálu sa pri bezdrôtových sieťach využíva najmä na vypočítanie vzdialenosti zariadenia, ktoré nám správu poslalo. V praxi je to možné aplikovať, keď chceme napr. pomocou určitej lateračnej metódy stanoviť pozíciu nejakého zariadenia.

Technika využíva **fyzikálnu vlastnosť degradácie signálu**, ktorý sa šíri v prostredí. Označuje sa **RSS** (*Received Signal Strength*) alebo **RSSI** (*Received Signal Strength Indication*). Pretože sila signálu, v závislosti od vzdialenosti, ktorú prekonáva, je menšia, môže byť meraná len na strane prijímateľa. Matematický model na určenie vzdialenosti `d` medzi odosielateľom a prijímateľom je vyjadrený týmto vzorcom 2.1.

$$PL(d) = PL(d_0) + 10 \times n \times \log(\frac{d}{d_0}) \tag{2.1}$$

kde `PL()` je funkcia straty signálu (*Path Loss*), meraná v decibeloch. Hodnota `n` je stratový exponent, ktorý udáva pomer veľkosti straty vzhľadom na vzdialenosť. Jeho hodnoty

sa bežne vyskytujú medzi *2* a *4*. Hodnota `d0` je referenčná vzdialenosť, určená z merania, kedy prijímač a vysielač sú v tesnej blízkosti. [15]

Nevýhodou tejto metódy je, že získané výsledky nie sú pri viacerých meraniach konštantné, práve naopak, môžu veľmi oscilovať, dokonca aj keď sa vysielač a ani prijímač nehýbe. Príčinou je nestálosť sily vysielacieho signálu a nestálosť životného prostredia. Rozsah zmien je v intervale $\pm 50db$. Do určitého stupňa je možné tento efekt neutralizovať opakovanými meraniami a filtrovaním nesprávnych chýb štatistickými technikami.

Druhým problémom je, že niektoré (najmä menej kvalitné) vysielače nie sú vždy kalibrované a pri tej istej sile signálu je možné získať rozličné RSSI hodnoty na rozličných zariadeniach.

Tretím problémom je prítomnosť prekážok v spojení s javom známym ako **MultiPath Fading**, ktorý je ilustrovaný na obrázku 2.3. Ak zariadenie získa odrazený lúč, bude výsledkom výpočtu väčšia vzdialenosť na rozdiel od skutočnej. V tomto prípade nepomôžu ani opakované merania. [12]

Tieto a iné problémy (viď kapitola 2.3.1) spôsobujú niekoľko chýb výpočtu. Na vyriešenie problému je nevyhnutné modelovať chybu signálového útlmu. Jedna možnosť je vloženie náhodnej premennej do `PL()` funkcie 2.2, kde X_ρ je **Gausova náhodná premenná** so štandardnou odchýlkou ρ [15].

$$PL(d) = PL(d_0) + 10 \times n \times \log(\frac{d}{d_0}) + X_\rho \tag{2.2}$$

Pri používaní RSSI metódy musíme vždy akceptovať a riešiť problémy spojené s prenosom signálu, ktoré spôsobujú chyby pri meraní vzdialenosti.

2.5 Topológia siete

Pri drôtových sieťach by mala mať topológia určitú, pevnú štruktúru. Najpoužívanejšie sú **prstencové**, **hviezdicové**, **zbernicové**, **stromové** a pod. V bezdrôtových sieťach neexistuje obmedzenie v počte topológii, ktoré môžu byť skonštruované. Napriek tomu existujú dva najčastejšie spôsoby tvorenia topológie [11]:

- Od bodu k bodu (Point-to-point) – Tiež nazývaný ad hoc mód. Tento postup zahŕňa minimum, kedy dve zariadenia komunikujú medzi sebou priamo, použitím toho istého sieťového štandardu. Výhodou je jednoduchosť. Je to neobmedzená topológia, ktorá nevyžaduje žiadnu infraštruktúru, napr. na smerovanie.

- Sieťový – Tento spôsob znamená vytvoriť ucelený sieťový prístup pre všetky uzly siete. Pri spojení s drôtovou sieťou musí existovať zariadenie – most, najčastejšie nazývaný **Basestation** alebo **Access point**.

Topológia bezdrôtovej siete môže byť matematicky definovaná ako množina aktívnych uzlov a množina aktívnych liniek. Matematicky je to možné vyjadriť grafom $G = (V, E)$, ktorý reprezentuje sieť. Potom V je množina všetkých uzlov a E je hrana, pričom $(v_1, v_2) \in E \subseteq V^2$, ak v_1 a v_2 môžu navzájom priamo komunikovať. Transformovaním získame výsledný graf $T = (V_T, E_T)$ pričom $V_T \subseteq V$ a $E_T \subseteq E$ [12].

2.5.1 Lokalizované topológie

V rámci zobrazenia topológie sa používajú aj lokalizačné techniky na zisťovanie polôh zariadení. Výsledný systém môže potom pomerne presne zobrazovať uzly s ich pozíciami prepojené úsečkami znázorňujúcimi vzájomné komunikačné kanály. Na lokalizáciu je možné použiť rôzne lateračné a angulačné metódy. **Angulációu** zistíme polohu uzlu tak, že získame uhol signálu, ktorý k nám dorazí, **laterációu** zo vzdialenosti uzlu. Obidve tieto techniky vyžadujú niekoľko tzv. referenčných uzlov, ktorých polohy sú známe. Na základe týchto uzlov môžeme zistiť polohy tých neznámych. Minimálny počet referenčných uzlov je tri, optimálne šesť.

Vzdialenosť je možné vypočítať niektorou z existujúcich metód na získanie vzdialenosti. Časom príchodu signálu **ToA** (*Time of Arrival*) ju môžeme zistiť, ak poznáme rýchlosť šírenia signálu. Ďalším príkladom je metóda **RSSI**, ktorá je predmetom kapitoly 2.4. [12]

2.5.2 Spravovanie topológie

V husto rozmiestnenej bezdrôtovej sieti, kde má uzol množstvo susedských uzlov, by priama komunikácia bola možná pri dostačujúco veľkej prenosovej sile. To by však vyžadovalo veľa energie, zaťažovalo by to MAC protokol a smerovacie protokoly by museli riešiť nestálosť siete, ak by sa uzly dokonca pohybovali [12]. Okrem pohybu je ďalším problémom aj vzdialenosť uzlov, s ktorou klesá energia a stúpa chybovosť pri prenosoch. Z toho dôvodu nie je tiež výhodné komunikovať s uzlom priamo.

Na vyriešenie týchto problémov je vhodné použiť **spravovanie topológie**. Cieľom tejto metódy je zmysluplne obmedziť množinu susedských uzlov a jednoznačne určiť komunikačnú cestu. Riešenie spočíva v kontrolovaní prenosovej sily vytvorením hierarchií v sieti alebo jednoduchým vypnutím niektorých uzlov.

Existuje mnoho techník na určenie komunikačnej cesty. Tie závisia najmä na technických prostriedkoch, ktoré bezdrôtová sieť poskytuje, predovšetkým na podpore multihop komunikácie. Dôležitým parametrami sú vzdialenosť a počet uzlov, cez ktorý budú pakety cez sieť putovať ku koncovému uzlu. Na každé poslanie paketu je potrebná určitá energia, ktorú musí každý uzol vynaložiť. Celková energia potrebná na poslanie jedného paketu cez niekoľko iných uzlov môže byť potom o dosť väčšia v porovnaní s priamym poslaním lenže s výhodou menšej chybovosti.

Netreba zabúdať aj na stavy, kedy chce uzol poslať správu uzlu, ktorý nie je v jeho dosahu, ale jeho prítomnosť je v sieti známa.

Spravovanie topológie býva často úzko spojené so smerovaním. Pomocou grafu potom ľahko usmerníme cestu paketov.

2.6 Senzorové siete

Senzorová sieť je infraštruktúra zložená zo snímajúcich, výpočtových a komunikačných elementov, ktoré umožňujú administrátorovi riadiť, pozorovať, reagovať na podnety a fenomény v rôznych životných prostrediach. *Administrátor* je typicky civilná, štátna alebo priemyselná osoba. *Životné prostredie* môže byť fyzický svet, biologický systém alebo architektúra informačných technológií a pod. [13] Príkladom môže byť továreň pozostávajúca z niekoľkých hál, ktoré vždy obsahujú malé kontrolné miestnosti. V nej sú zobrazované rôzne informácie o stave haly (stav rozvodových uzáverov, náradia, teplota a tlak skladovaných materiálov, atď.). Senzory, ktoré ich zaznamenávajú bývajú väčšinou lacné v porovnaní so

zabezpečenými rozvodmi použitej káblovej siete. Preto je vhodné ju nahradiť bezdrôtovou technológiou. [8]

Sledované prostredie býva často určitým spôsobom obmedzené, napr. vysokou nadmorskou výškou, takže snímajúce zariadenia musia mať väčšinou vlastný zdroj energie a pod. Zmeny v prostrediach bývajú veľmi zriedkavé, takže požiadavky na dáta prenášané prostredníctvom siete sú tiež nízke. Lenže spoľahlivosť musí byť vysoká. Bezdrôtové senzorové siete pozostávajúce z množstva uzlov, poskytujúce viacnásobné smerovanie a multihop komunikáciu, môžu spĺňať tieto požiadavky [6].

Bezdrôtové senzorové siete sú vnímané ako dôležitá technológia, ktorá by mala v niekoľkých nasledujúcich rokoch obohatiť množstvo aplikácii [13].

2.6.1 Použiteľnosť v aplikáciách

Senzorové siete môžu pozostávať z množstva rôznych typov senzorov, ako sú *seizmické*, *magnetické*, *termálne*, *vizuálne*, *infračervené*, *akustické* a *radarové*, ktoré môžu monitorovať rôzne vplyvy životného prostredia (napr. tabuľka 2.6.1).

Podmienky životného prostredia
Teplota
Vlhkosť
Pohyb
Svetlo
Tlak
Zloženie pôdy
Vlhkosť vzduchu

Tabulka 2.1: Príklady vplyvov životného prostredia [4]

Môžu byť použité k nepretržitému snímaniu, zachyteniu udalosti, identifikácii udalosti, lokálnemu snímaniu alebo kontrole činiteľov a pod. Koncept mikrosnímania a bezdrôtového spojenia zariadení dáva prísľub viacerým oblastiam aplikácii, ako sú vojenské, pre životné prostredie, zdravotné, do domácnosti, komerčné, prieskumné, chemické a na zachytávanie katastrof. [4]

Kapitola 3

Systém WSageNt

WSageNt je agentný systém, ktorý bol vytvorený ako projekt diplomových a bakalárskych prác pod vedením pána Ing. Františka Zbořila Ph.D. Jeho základom je bezdrôtová senzorová sieť zložená zo senzorových zariadení MICAz a IRIS schopných zachytávať zmeny okolia a komunikovať navzájom. Zariadenia sú naprogramované programovacím jazykom **NesC** a pracujú pod operačným systémom **TinyOS**. Správanie agenta je definované jazykom **ALLL**, ktorý bol pre tieto účely tiež vytvorený.

3.1 Zariadenie MICAz od firmy CrossBow

Firma CrossBow sa venuje tvorbe bezdrôtových senzorových technológií od roku 1995. Vo svojej ponuke obsahuje množstvo senzorových zariadení, označovaných **mote**. V tomto texte budeme tento názov tiež používať[1].

Ako senzorové zariadenie – uzol siete, bol v projekte použitý mote **IRIS** a **MICAz**. MICAz je posledný produkt z generácie od firmy CrossBow. Jeho základné parametre sú [3]:

- Disponuje senzorovou doskou s označením MPR2400CA.
- Obsahuje procesor Atmel ATMega128L @ 8 MHz.
- Obsahuje RF vysielač podliehajúci štandardu IEEE 802.15.4.
- Vysielacia frekvencia je 2.4 – 2.48 GHz.
- Vysiela rádiové spektrum rezistentné voči interferenciám.
- Prenosová rýchlosť rádia je 250 kbps.

Dôležitým zariadením v sieti je **basestation**, ktorý sieťovo prepojuje počítač s motes. Každá MICAz mote môže pracovať ako basestation, keď je pripojená k počítaču. Firma CrossBow vyrába pre tento účel zariadenia s označením MIB510, MIB520 a iné. Tie poskytujú sériové alebo USB pripojenie pre programovanie alebo komunikáciu s motes. [3]

[1] note pre jednotné číslo, motes pre množné

3.2 TinyOS, NesC

TinyOS je operačný systém použitý pri vývoji systému WSageNt. Je nenáročný na zdroje a od verzie 1.1.7 je podporovaný aj pre platformu MICAz. Vo väčšine prípadov ponúka platformovo nezávislý vývoj aplikácii s množstvom dostupných, štandardných knižníc. Tie je možné použiť pre platformu MICAz (ale aj pre iné, napr. Imote2). Okrem nich je možné použiť aj špeciálne, hardvérovo-nezávislé knižnice. TinyOS je optimalizovaný s ohľadom na využitie zdrojov, hlavne pamäti RAM. Jeho programovacím jazykom, je jazyk podobný jazyku C, s názvom NesC.

Pri preklade sú všetky použité knižnice spojené s kódom programu, ktorý je vytvorený užívateľom, a vložené do binárneho súboru. Ten je následne možné vložiť do zariadenia. Všetky adresy premenných a funkcií sú pevne stanovené a nemôžu byť zmenené, čo platí aj pre hardvérové komponenty. Napríklad je presne určené z akej adresy sa čítajú dáta a pod.

Kompiláciu programu je možné zabezpečiť vytvoreným súborom *Makefile*, ktorý aktivujeme štandardne pomocou programu *make*. Ten zariadi spojenie lokálneho makefilu a makefilu pre špecifickú platformu.

Pre platformu MICAz existuje simulátor s názvom **TOSSim**, pomocou ktorého je možné simulovať spustenie vytvoreného programu na PC. Je možné simulovať viac uzlov súčasne, vrátane kvality spojenia a šumu pri prenose dát. [9]

3.2.1 Basestation

Basestation je základná TinyOS utilita, ktorá sa správa ako most medzi sériovým portom a bezdrôtovou sieťou. Keď získa paket zo sériového portu, pošle ho rádiovým signálom do siete a naopak. Pretože TinyOS vytvorila aplikáciu, ktorá generuje a posiela pakety cez sériový port do mote, je možné použitím basestation komunikovať priamo cez počítač s daným mote v sieti. Ak pripojíme mote priamo k počítaču, bude sa správať ako basestation. Môžeme tak urobiť pomocou sériových portov COM, USB a pod.

TinyOS obsahuje niekoľko knižníc, pomocou ktorých je možné komunikovať s basestation. Sú to aplikácie napísané v jazyku Java. Jedným z nich je program Listen, ktorý zobrazuje všetky pakety prichádzajúce z portu. Na OS Windows ho spustíme:

```
java net.tinyos.tools.Listen -comm serial@COM1:micaz
```

Na Unixových systémoch:

```
java net.tinyos.tools.Listen -comm serial@/dev/ttyS0:micaz
```

Príkaz hovorí o použití sériového portu, ktorý sa nachádza na `dev/ttyS0` s automaticky správnou rýchlosťou pre MICAz mote. Parameter `--comm` určuje zdroj k sériovému portu. Pridaním `:19200` stanovíme prenosovú rýchlosť na 19200 baudov. [20]

3.3 ALLL (Agent Low Level Language)

V tejto podkapitole stručne zhrniem jazyk ALLL, ktorý bol vytvorený za účelom prenosu správ medzi zariadeniami mote. Síce je veľkou súčasťou systému WSageNt, webové rozhranie s ním nie je úzko spojené. Nevytvára ho a ani ho nemodifikuje pri komunikácií. Je len jej obsahom. Znamená to, že je ho možné nahradiť, bez narušenia funkčnosti webového rozhrania.

ALLL je založený na teoriach o **BDI** (*Belief, Desire, Intention*) agentoch. Tie však nie sú predmetom tohto textu. Viac je možné sa dozvedieť v práci p. Horáčka [9] a p. Spáčila [18].

3.3.1 Základné funkcie

Jazyk ALLL poskytuje popis agenta. Tento popis je predstavovaný jeho plánmi, ktoré sú tvorené konkrétnymi akciami. Kód agenta je rozdelený do 7 častí a to: **BeliefBase** (báza znalostí), **PlanBase** (báza plánu), **Plan** (zámer), **InputBase** (datový sklad), **Register 1**, **Register 2** a **Register 3**. Registry 1, 2 a 3 sú pomocné a môžu byť použité na rôzne výpočty a iné funkcie. Každá časť obsahuje určité akcie, ktoré je možné hierarchicky zanorovať. Tabuľka 3.3.1 obsahuje príklad agenta, ktorý obsahuje akcie: `príjem` na počúvanie a `blik` na prepínanie LED diód. Je umiestnený na uzle číslo *3* a komunikuje s agentom na uzle č. *2*. Je možné vidieť, že akcie sa opakujú – prebiehajú v slučke, čím sa zaistí, že agent bude neustále počúvať.

Časť	Obsah
PlanBase	`(prijem,($(s)&(2)?(2)$(a)!(2,&2)^(blik)))(blik,(&(1)$(r,&2)&(3)$(f,&1)$(l,&3)&(2)$(r,&1)&(1)$(f,&2)$(w,&1)$(l,&3)^(prijem)))`
Plan	`^(prijem)`

Tabulka 3.1: Ukážka kódu agenta – počúvajúci agent [9]

3.3.2 Získanie RSSI

Keďže jazyk ALLL nám umožňuje prenášať rôzne formy dát, napr. dáta nasnímané zo senzoru a pod., môžme pomocou neho posielať aj namerané RSSI. Tie sa budú nachádzať v báze znalostí, do ktorej si agent uchováva konkrétne hodnoty zistené počas svojej činnosti. Na to aby sme ich získali potrebujeme agenta, ktorý spustí požadovanú akciu. Tabuľka 3.3.2 obsahuje agenta, v ktorého pláne sa nachádza akcia $(n) pre získanie sily signálu k svojím okolitým susedom. Po spustení akcie sa požadované hodnoty uložia v tvare (NB, adresa_zariadenia, sila signálu). Za touto akciou sa nachádza ešte príkaz agentnej mobility, ktorý hovorí, že sa má agent poslať na určitý uzol, v tomto prípade s adresou *1*, čo je adresa basestation. Agent sa prenesie aj s hodnotami RSSI. Tabuľka 3.3.2 obsahuje údaje od dvoch mote s adresami *1* a *3* a silou signálu *12* a *6*.

Časť	Obsah
Plan	(n)(m,(1))
BeliefBase	(NB, 1, 12)(NB, 3, 6)

Tabulka 3.2: Ukážka obsahu báze znalostí po získaní RSSI [9]

Hodnota sily signálu, ktorú získame sa bude pohybovať v rozmedzí 0 až 255, pričom 0 znamená nulovú silu a 255 maximálnu silu signálu. Je to prepočítaná hodnota nezávislá od technológie použitých zariadení.

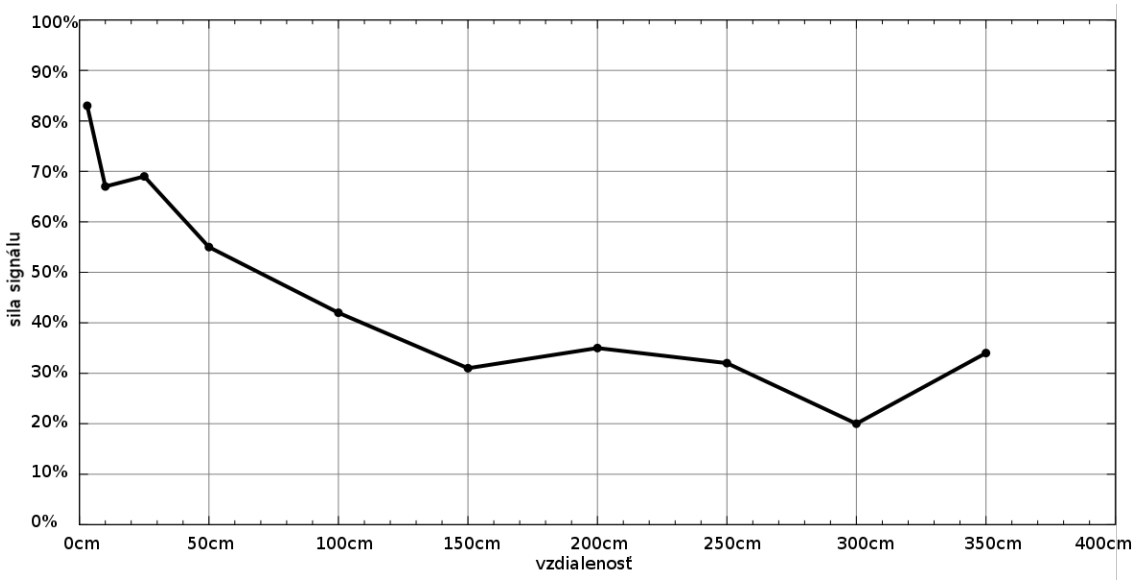
3.4 Experimentálne meranie sily signálu

V tejto podkapitole sa pozrieme na reálnu silu signálu, ktorú merajú motes od Crossbow. Použijeme dva motes so označením IRIS a pri rôznych vzdialenostiach si necháme získať silu signálu. Meranie vykonáme v miestnosti pri priamej, vzájomnej viditeľnosti obidvoch zariadení, vrátane basestation, ktorá sa bude nachádzať pri jednom z nich. Pretože skutočnú silu meranú v decibeloch nepoznáme, výsledky sú prepočítané na **percenta maximálnej sily** získanej z intervalu 0 až 255 (viď sekcia 3.3.2). Môžeme ich vidieť v tabuľke 3.4.

Bolo nameraných celkovo päť meraní, pričom každé z nich obsahuje hodnoty získané z oboch motes. Pre štatistické účely vypočítame aritmetický priemer μ a smerodajnú odchýlku σ.

l [cm]	1.1 [%]	1.2 [%]	2.1 [%]	2.2 [%]	3.1 [%]	3.2 [%]	4.1 [%]	4.2 [%]	5.1 [%]	5.2 [%]	μ [%]	σ [%]
3	80	80	84	80	84	80	84	84	84	84	83	4,42
10	67	67	67	67	70	70	67	63	67	63	67	5,69
25	70	70	67	67	70	67	70	70	70	67	69	4,42
50	49	49	56	56	56	56	56	56	56	56	55	7,21
100	49	49	28	46	46	46	39	39	42	39	42	15,59
150	28	25	35	25	31	28	28	39	31	42	31	14,20
200	39	39	42	35	28	28	31	35	35	39	35	11,73
250	31	31	35	21	25	31	35	39	39	31	32	12,40
300	31	31	31	28	31	21	28	31	14	0.5	20	26,67
350	31	31	31	28	39	39	35	35	39	35	34	8,82

Tabulka 3.3: Experimentálne meranie sily signálu. Pomer vzdialenosti k sile signálu. **l** – vzdialenosť, μ – aritmetický priemer, σ – smerodajná odchýlka.



Obrázek 3.1: Graf merania priemernej sily signálu.

V tabuľke je možné vidieť premenlivosť sily signálu. Dokonca v určitých prípadoch

môžeme pozorovať veľké odchýlky od priemernej hodnoty. Zaujímave je, že pri veľmi malej vzdialenosti sú výsledky pomerne nízke a pri žiadnom z meraní nebola sila signálu väčšia ako 90% z maximálnej. Ak by sme pokračovali v zväčšovaní vzájomnej vzdialenosti, môžme predpokladať, že signál bude slabnúť.

Jednotlivé hodnoty priemernej sily signálu sme premietli do grafu, ktorý je znázornený na obrázku 3.1. X-ová os je reprezentovaná vzdialenosťou a Y-ová percentuálnou, priemernou silou signálu. Môžeme vidieť, že sila skutočne so zväčšujúcou vzdialenosťou klesá, aj keď pri pomerne veľkej chybe merania.

Lepšie výsledky by sme mohli dosiahnuť pri väčšom počte meraní a s väčšou škálou vzdialeností. Podstatný je fakt, že sila signálu je veľmi premenlivá a veľmi záleží od prostredia, v ktorom sa zariadenia nachádzajú a od rôznych iných faktorov.

3.5 Popis platformy a jej služby

Platforma disponuje niekoľkými základnými službami, ktoré sú [9]:

- **Výstup na LED diódy**, ktoré obsahuje každý mote, je možné zapínať, vypínať alebo len spôsobiť bliknutie. Sú tri, vo farbách: červená, zelená a žltá. Používajú sa najmä na indikáciu svojho stavu alebo činnosti pre užívateľa.
- **Služby pre riadenie uspania**, medzi ktoré patria:
  1. Uspanie na požadovanú dobu. Touto službou pozastavíme činnosť interpreta na požadovanú dlhú dobu danú v milisekundách.
  2. Čakanie na prichádzajúcu správu.
  3. Ukončenie činnosti agenta. Interpret volá tuto službu v ojedinelých prípadoch, kedy si s istotou želá ukončiť svoju činnosť.
- **Služby pre získavanie dát zo senzoru.** V motes MICAz je použitá senzorová doska MPR2400CA, pre ktorú ovládač chýba, takže nebolo možné využiť všetky služby, ktoré poskytuje. Bol použitý ovládač *basicsb*, ktorý umožňoval použiť služby:
  1. Získanie aktuálnej teploty zo senzoru.
  2. Intervalové meranie, využitím časovača. Všetky hodnoty sa zapisujú do histórie.
  3. Virtuálny modul a rozhranie pre simuláciu flash.
  4. Služby využívajúce históriu dát, medzi ktoré patrí napr. výpočet priemernej teploty za určitý časový interval. Na základe týchto hodnôt je možné potom vyhodnotiť ďalšiu akciu.
- **Služby pre prácu so zoznamami**, ktoré boli implementované ako obdoby operátorov cad a cdr v jazyku LISP.
- Odoslanie a príjem správ.

Služby pre odoslanie a príjem správ pracujú s paketmi, ktoré cestujú v sieti. Každý paket má určitú štruktúru, ktorá musí byť dodržaná v rámci vytvoreného protokolu. Protokol takisto definuje ako prebieha riadenie odosielania a príjmu správ. Každý paket obsahuje svoje poradové číslo, ktoré je umiestnené hneď na začiatku. Za ním nasledujú smerové informácie, a to zdrojová a cieľová adresa. Za nimi nasleduje veľkosť dátovej časti, tzn.

dátových paketov, ktoré túto veľkosť musia dodržať. Poslednými parametrami sú začlenenie do skupiny paketov a identifikátor typu správy. Hlavička je prvý paket, ktorého číslo je 0. Za ním nasledujú dátové pakety, pričom každý nasledujúci má poradové číslo o 1 väčšie.

Pri odosielaní správy nastavíme časovač a čakáme, kým nám opačná strana nepošle potvrdenie o prijatí paketu. Ak potvrdenie nepríde včas, skúšame poslať opäť ten istý paket.

Pri prijímaní správy príjemca čaká na paket s číslom 0. Keď príde, komunikuje len s daným odosielateľom. Ak mu v tom čase príde paket od iného odosielateľa, zahodí ho.

Špecifickým typom odosielanej správy je kód samotného agenta. Tomuto prenosu v systéme hovoríme agentná mobilita. Spoznáme to podľa identifikátoru typu správy, ktorý sa nachádza na konci v každej hlavičke. [9]

štart	Cieľ. Adr.	Zdroj. Adr	Dĺžka dát	Skupina	Typ	Var1	var2
00	00 02	00 01	04	22	06	8F 43	66 A0

Tabulka 3.4: Príklad hlavičky s dátmi [9]

Kapitola 4

Webové rozhranie v Java EE

V tejto kapitole stručne charakterizujeme tvorbu webových rozhraní v `Java EE`. Táto platforma nám poskytuje veľké množstvo riešení v oblastí webových informačných systémov. Podporuje niekoľko technológií, pomocou ktorých sú systémy výkonné, stabilné, bezpečné a pod. Sú to najmä *JavaBean*, *Java Servlets*, *JavaServer Pages*, *Java Persistence*, *JavaServer Faces* a mnohé ďalšie. Nie je dôležité sa danými technológiami zaoberať podrobne, pretože sú len nástrojom na vytvorenie mechanizmov, ktoré sú podstatnejším predmetom tejto práce. Okrem toho, samotná Java EE už bola témou mojej bakalárskej práce, v ktorej je môžné sa dočítať viac[1].

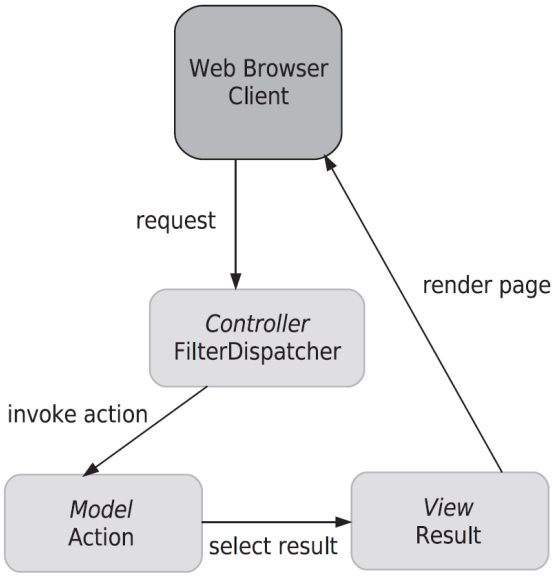
4.1 Struts 2

Struts 2 je druhou verziou open–source frameworku určeného pre vývoj webových aplikácií pod platformou Java EE. Vysoká úroveň tohto frameworku používa zaužívaný návrhový vzor **MVC** pre webové technológie. V Struts 2 sú jednotlivé zložky implementované ako **akcie** (*action*) pre model, **výsledok** (*result*) pre pohľad (*view*) a **FilterDispetcher** ako kontrolór (*controller*). Princíp pracovania zobrazuje obrázok 4.1. Pozrime sa na jednotlivé zložky zvlášť.

FilterDispatcher. Pri webových aplikáciach dáva zmysel hovoriť najprv o tejto zložke, pretože je prvou komponentou v procese komunikácie klienta so serverom. Jeho úlohou je mapovanie klientských požiadaviek na akcie. V Java EE sú za požiadavky zodpovedné objekty – tzv. servlety. FilterDispetcher plní úlohu servlet filtra, ktorý dozerá na každú prichádzajúcu požiadavku, na základe ktorej rozhodne, ktorá akcia sa má vykonať. Našou úlohou je už len ho informovať o tom, ktorá URL adresa má byť namapovaná na ktorú akciu. To je možné urobiť jednoducho napr. pomocou XML konfiguračných súborov: [5]

```
<struts>
  <package name="default" namespace="/" extends="struts-default">
    <action name="Register" class="cz.vutbr.wsagent.controlpanel.Register"
      <result name="success">/RegisterSuccess.jsp</result>
      <result name="input">Register.jsp</result>
    </action>
  </package>
</struts>
```

[1] viac na [14])



Obrázek 4.1: Princíp pracovania frameworku Struts 2 [5]

Model je v podstate tzv. *čierna skrinka*, ktorá obsahuje jadro všetkých požadovaných funkcií. Technicky povedané, že je vnútorným stavom aplikácie. Tento stav je zložený z oboch prvkov systému – dátového modelu a aplikačnej logiky. Všetko ostatné je len GUI alebo prepojenie. Objekt action, ktorý model implementuje má teda 2 základné úlohy. Prvou je zapuzdrenie všetkých volaní aplikačnej logiky do jednej výkonnej jednotky. Po druhé slúži ako miesto pre dátové prenosy.

Ako je vidieť na obrázku 4.1, po tom čo kontrolór získa HTTP požiadavku, musí zhodnotiť jej mapovanie a určiť, ktorá z akcií by mala obslúžiť požiadavku. Keď nájde správnu akciu uvedie ju do činnosti. Tento vyvolávací (*invocation process*) proces vykonaný kontrolórom splní obe úlohy – pripraví potrebné dáta a zavolá aplikačnú logiku danej akcie. Keď spracovanie akcie je ukončené, príde tá správna doba na zobrazenie výsledku užívateľovi, ktorý poslal požiadavku. Predaním výsledku príslušnej Struts2 komponente pre pohľad končí jej práca. [5]

Model je tvorený prevažne triedami, ktoré sa nachádzajú v balíčkoch podľa určitej architektúry, napr. vrstvenej. Okrem toho Struts2 obsahuje rozhrania určené na spracovanie požiadaviek, ktoré je vhodné implementovať.

Pohľad je prezentačnou komponentou v MVC návrhovom vzore. Ak sa pozrieme opäť na obrázok 4.1, vidíme, že výsledok vráti JSP stránku klientovi. Táto stránka je užívateľské rozhranie, ktoré reprezentuje stav aplikácie užívateľovi. To je hlavná úloha tejto komponenty. Užívateľ s ňou potom môže interagovať [5].

Použitím frameworku sa prezentačné stránky nijak od tých ostatných JSP stránok nemusia odlišovať. Môžeme použiť zaužívané techniky JSP skriptovania. Struts nám však poskytuje balík logiky a predformátovaných elementov, ktorý zahrnieme do stránky ako štandardnú značkovaciu knižnicu, takto:

```
<\%@ taglib prefix="s" uri="/struts-tags" \%>
```

Následne môžeme využívať elementy, ktoré sa vyznačujú prefixom `<s:`.

4.1.1 Implementácia akcií

Struts 2 framework obsahuje niekoľko rozhraní, ktoré sú určené pre implementovanie akcií užívateľom. Tie poskytujú množstvo prostriedkov za nízke výpočtové náklady. Najčastejšie akcie vytvoríme pomocou rozhrania `com.opensymphony.xwork2.Action`. Toto rozhranie definuje len jednu metódu `String execute() throws Exception`. Význam tejto metódy spočíva vo vykonaní aplikačnej logiky, tzn. zavolanie všetkých mechanizmov pre získanie všetkých potrebných dát, ktorými naplní danú prezentačnú komponentu. Ak v nastaveniach akcie nie je dané inak, je táto metóda volaná automaticky. Nie je výhodné implementovať rozhranie `Action` len kvôli tejto metóde, ďalšou vymoženosťou sú reťazcové konštanty:

```
public static final String ERROR "error"
public static final String INPUT "input"
public static final String LOGIN "login"
public static final String NONE "none"
public static final String SUCCESS "success"
```

Tieto konštanty sú použité ako kontrolné reťazcové hodnoty, ktoré sú vrátené metódou `execute()`. Výhodou je, že tiež sú používané vnútorne frameworkom struts. Prínosom je jednak zjednotenie výsledkov ale aj inteligentnejšie zaobchádzanie s akciami. Výsledok úspešnej akcie deklaruje konštanta `SUCCESS`, zatiaľ čo výsledok neúspešnej akcie definuje `ERROR` a pod. [5]

Trieda ActionSupport

Najpoužívanejšou implementáciu rozhrania `Action` je trieda `ActionSupport`, ktorá rozširuje rozhranie o niekoľko možností, ako je validácia a lokalizácia chybových správ. Tie sú poskytované kombináciou zachytávačov (viď kapitola 4.1.3) a rozhraním. Zachytávače kontrolujú vykonávanie služieb, zatiaľ čo metódy, ktoré implementujú rozhranie, sú vyvolané zachytávačmi.

Ak užívateľ rozširuje (*extends*) triedu `ActionSupport`, a zároveň jeho balík akcií rozširuje základný balík `struts--default`, môže využiť základnú validáciu vstupných dát (najčastejšie formulárov). To umožňuje zachytávač `DefaultWorkflowInterceptor` z balíku `struts--default`. V okamihu keď je uvedený do činnosti vyhľadá metódu `validate()`. V nej je užívateľom implementovaná logika na validovanie dát. Táto metóda je definovaná v `com.opensymphony.xwork2.Validateable`, ktorú trieda `ActionSupport` implementuje s prázdnym obsahom. Pri použití ju musíme preťažiť a vložiť validáčnú logiku. [5] Príkladom môže byť overenie, či užívateľ vyplnil registračné údaje pre meno a heslo vo formulári.

```
public void validate(){

    if ( getUsername().length() == 0 ){
        addFieldError( "username", "Username is required." );
    }

    if ( getPassword().length() == 0 ){
        addFieldError( "password", getText("password.required"));
    }
}
```

Predpokladajme existenciu privátnych atribútov *username* a *password*, ktoré sú podporované verejnými *get* a *set* metódami. Pri zistení, že dĺžka niektorého poľa je nulová, použijeme metódu `addFieldError()` pre výpis chyby užívateľovi. Po tom, čo validácia skončí, je riadenie opäť vrátené zachytávaču *Workflow*. Ten sa pozrie, či boli generované nejaké chybné správy. Ak áno, okamžite zruší spracovanie požiadavky a vráti všetky chybné správy do vstupného formuláru, kde sa zobrazia. Prvý parameter metódy addFieldError() predstavuje kľúč vstupného textového pola. Druhý parameter je chybová správa, ktorá má byť pri ňom vypísaná. [5]

Validácia vstupných údajov

Ďalšou výhodou je implementácia rozhrania `com.opensymphony.xwork2.TextProvider`, ktorá poskytuje prístup k textovým správam – oznamom. Metódy sú schopné získať správy zo súborov nazvaných `NázovTriedy.properties`. Ku konkrétnej správe sa pristupuje pomocou kľúča, ktorý použijeme ako parameter metódy `getText()`. Táto metóda je jedna z mnohých na získanie príslušnej správy. V balíku, kde sa nachádza trieda *Register* sa musí vyskytovať aj súbor `Register.properties`. Ten jednoducho obsahuje priradenie danej správy: `password.required=Password is required.`.

Trieda *ActionSupport* ďalej obsahuje jednoduché riešenie pre podporu rôznych jazykov. Rozhranie `com.opensymphony.xwork2.LocaleProvider` poskytuje metódu `getLocale()`, ktorá vráti správu v konkrétnom jazyku v závislosti od nastavení poslaných od internetového prehliadača.

Ďalším typom validácie je pomocou metadatových `xml` súborov, ktoré definujú validačnú logiku. Tie môžu nahradiť metódu `validate()`. Výhodou je, že poskytujú implementované základné funkcie, ako sú kontrola na prázdny vstup, dĺžku vstupu, kontrola formátu dátumu a času, emailu, url adresy, intervalu číselných hodnôt, kontrola regulárnymi výrazmi a pod. Špecifickejšie validácie si musí vývojár vytvoriť sám rozšírením základnej triedy `FieldValidatorSupport`.

Tak ako aj v predchádzajúcom príklade sme od užívateľa požadovali neprázdne vstupné údaje, môžme tak urobiť aj pomocou súboru `Register-validation.xml`, ktorý obsahuje nasledujúci kód:

```
<validators>
     <field name="username">
          <field-validator type="requiredstring">
               <message>You must enter a value for username.</message>
          </field-validator>
     </field>
     <field name="password">
          <field-validator type="requiredstring">
               <message>You must enter a value for password.</message>
          </field-validator>
     </field>
</validators>
```

Všeobecne, ku každej triede, ktorá má byť validovaná týmto spôsobom, musí existovať súbor s príponou `-validation.xml`.

Ďalším typom kontroly vstupných údajov je použitie anotácií. Tento princíp je v zásade rovnaký a umožňuje rovnakú validačnú silu. Nasledujúci príklad opäť vyžaduje od užívateľa

zadanie mena. Viac o tom nájdete na [5].

```
@RequiredStringValidator(type = ValidatorType.FIELD,
     message = "Username is required.")

public String getUsername() {
     return username;
}
public void setUsername(String username) {
     this.username = username;
}
```

Metóda `execute()`

Po bezchybnom validovaní vstupných údajov sa dostáva na rad metóda `execute()`. V nej je vykonávaná požadovaná aplikačná logika. Príkladom môže byť vytvorenie nového užívateľa systému a jeho uloženie do databázy. Návratová hodnota tejto metódy deklaruje výsledok akcie. V tomto prípade skončila akcia úspechom a systém môže zavolať príslušnú prezentačnú komponentu, ktorá slúži na zobrazenie pohľadu, ktorá je v konfiguračnom súbore určená pre tento účel. [5]

```
   public String execute(){
      User user = new User();
      user.setUsername( getUsername() );
      user.setPassword( getPassword() );
      /* save the object to the database */
      new UserDAO().saveToDatabase(user);
      return SUCCESS;
}
```

4.1.2 OGNL (*Object-Graph Navigation Language*)

OGNL je výkonná technológia, ktorá bola integrovaná do frameworku Struts 2 na pomoc s dátovými prenosmi a typovou konverziou. Je to prostriedok medzi HTTP vstupom, výstupom a vnútorným riadením Javy. V zásade ide o mapovanie parametrov metódy GET a POST (HTTP požiadavky a odpovede) na určité atribúty v Java triedach, ktoré implementujú dané akcie. Predpokladajme, že naša JSP stránka obsahuje tento nadpis:

```
<h1>User: <s:property value="username" />, Welcome!</h1>
```

Za meno užívateľa bude dosadená hodnota z atribútu `username` z ukážkovej triedy Register. Každý takýto atribút musí byť vo forme *JavaBeans*, tzn. minimálne musí obsahovať `get` a `set` metódy pre získanie a nastavovanie jeho hodnoty:

```
private String username;

public String getUsername() {
     return username;
}
```

```
public void setUsername(String username) {
     this.username = username;
}
```

V prípade, že chceme pomocou formulára odosielať dáta na server alebo použiť parametre HTTP požiadavky, musí daná trieda obsahovať atributy. Tie potom budú už pri validácií obsahovať odoslané hodnoty.

Okrem primitívnych typov atribútov OGNL podporuje aj typy odvodené od základnej triedy `Object`. K atribútom objektových typov sa potom pristupuje hierarchicky, tzn. ak chceme v JSP stránke pristúpiť k menu užívateľa, ktorý je reprezentovaný JavaBean triedou `User`, zadáme `<s:property value="user.username" />`.

Niektoré elementy JSP dokumentu nepovoľujú vkladanie hodnôt atribútov pomocou elementu `<s:property>`. Okrem toho niekedy potrebujeme získať dáta pomocou špeciálneho výrazu. OGNL pre tento prípad obsahuje konštrukciu `%expression`, ktorá signalizuje frameworku, že sa nejedná o reťazec znakov, ale o špeciálny výraz, ktorý je vyhodnotený.

Parametre, ktoré sa pomocou OGNL z HTTP požiadavok a odpovedí konvertujú na atribúty a naopak, sú uložené v lokálnom sklade, nazvanom `ValueStack`.

Struts 2 framework obsahuje vstavanú podporu pre konvertovanie HTTP reťazcov znakov a nasledujúcich Java typov:

- `String`.
- `boolean/Boolean` – ak sú reťazce `true` a `false`.
- `int/Integer`, `float/Float`, `long/Long`, `double/Double`.
- `Date`.
- `array` – Každý element musí byť schopný konvertovať do typu akého je pole definované.
- `List` – Implicitne naplnený textovými objektami.
- `Map` – Implicitne naplnený tiež textovými objektami.

Znakové reťazce primitívnych typov môžu byť konvertované aj ako primitívne typy, aj ako objekty.[5]

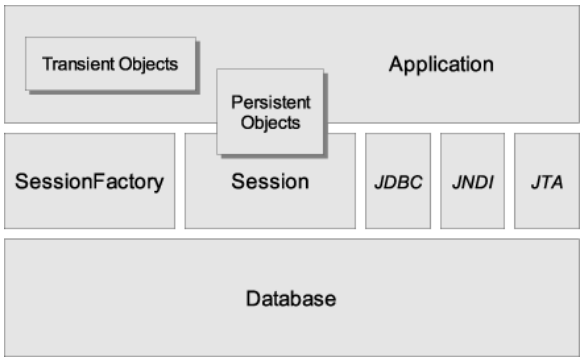
4.1.3 Zachytávač (Interceptor)

Vyvolanie a vrátenie sa z každej akcie musí prejsť cez sadu komponent, ktoré sa nazývajú zachytávače (*interceptors*). Je to dôležitý proces vo frameworku Struts 2, pretože každá výnimka, ktorá môže v systéme nastať a ak môže byť ošetrená na nižšej úrovni ale nie je, je odchytená týmito komponentami. Niektoré zo zachytávačov pracujú len pred vyvolaním určitej akcie, iné naopak len po skončení akcie. Dôležitou črtou je, že poskytujú jednotnú deklaráciu úloh v sebestačných, znovu-použiteľných komponentách odlúčených od samotných akcií. [5]

4.2 Hibernate

Takmer každý informačný systém potrebuje nástroj na prácu s perzistentnými dátmi. Výhodným spôsobom pre webové technológie je použiť relačnú databázu. My však chceme uložiť priamo vytvorené inštancie objektov. **ORM** (*Object Relational Mapping*) je programovacou technikou, ktorá nám to umožní. Je to prevádzanie objektov na relačné dotazy rôznych typov databáz. Hibernate je knižnica poskytujúca framework, ktorý implementuje ORM. Zároveň dodržiava **JPA** (*Java Persistence API*), takže je aj oficiálnou implementáciou tohto abstraktného rozhrania. Dôležitou vlastnosťou je zachovanie perzistencie objektov. Samozrejme, vrámci objektovo-orientovaného programovania, ďalej podporuje dedičnosť, polymorfizmus a asociačné, kompozičné a zovšeobecnovacie väzby. Veľkou výhodou frameworku hibernate je vysoká abstrakcia práce s databázou, ktorá umožňuje použitie niekoľkých rôznych databázových systémov a ak sa užívateľ rozhodne pre iný, je to len otázka nastavenia konektoru k databáze. Túto abstrakciu podporuje aj vlastný dotazovací jazyk **HQL** (*Hibernate Query Language*) [7].

Architektúra Hibernate sa skladá z viacerých vrstiev, kde každá vrstva využíva služby tých nižších.



Obrázek 4.2: Hibernate architektúra [7]

Obrázok 4.2 znázorňuje jednoduché rozdelenie, ktoré ju popisuje:

- **SessionFactory** (`org.hibernate.SessionFactory`) je to továreň na vytvorenie klienta pre pripojenie do databázy. Voliteľne môže udržovať cache data, ktoré sú znovupoužitelné pri transakciach.
- **Session** (`org.hibernate.Session`), ďalej ako sedenie, je objekt s krátkou životnosťou reprezentujúci konverzáciu medzi aplikáciou a databázou. Obsahuje pripojenie do JDBC a je továrňou pre objekt `Transaction`. Session udržuje povinné cache data perzistentných objektov, ktoré sú použité pri navigovaní alebo vyhladávaní objektov.
- **Perzistentné objekty** (*Persistent Objects*) sú jednoduché objekty aplikačnej logiky obsahujúce perzistentný stav. Sú asociované k presne jednému sedeniu (Session). Ak je sedenie ukončené, sú oddelené a voľne dostupné k použitiu na ľubovoľnej aplikačnej vrstve.
- **Krátkodobé objekty** (*Transient Objects*) sú inštancie perzistentných tried, ktoré nie sú asociované k žiadnemu sedeniu.

- **Transaction** (`org.hibernate.Transaction`) je objekt používaný na špecifikovanie atomických operácií. Vytvára abstrakciu aplikačnej logiky, ktorá využíva **JDBC**, **JTA** alebo **CORBA** tranzakcie, ktoré sa nachádzajú na nižšej vrstve.
- **ConnectionProvider** (`org.hibernate.connection.ConnectionProvider`) je továreň JDBC pripojení. Abstrahuje aplikačnú logiku nižších tried DataSource alebo DriverManager.
- **TransactionFactory** (`org.hibernate.TransactionFactory`) je továreň pre inštancie triedy Transaction. [7]

4.2.1 Konfigurácia

Prvým krokom je nastavenie samotného hibernate a nastavenie JDBC pripojenia do databáze a s tým súvisiacich parametrov. Všetky nastavenia prebiehajú v konfiguračných xml súboroch pre jednoduchú manipuláciu. Príkladom je nasledujúci kód:

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">
       org.hibernate.dialect.MySQLDialect
    </property>
    <property name="hibernate.connection.driver_class">
       com.mysql.jdbc.Driver
    </property>
    <property name="hibernate.connection.url">
       jdbc:mysql://localhost:3306/wsagent
    </property>
    <property name="hibernate.connection.username">
       root
    </property>
    <property name="hibernate.connection.password">
       root
    </property>
    <property name="hibernate.hbm2ddl.auto">
       validate
    </property>
    <mapping resource="cz/vutbr/wsagent/db/hibernate/agent.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Týmto spôsobom nastavujeme parametre tried `org.hibernate.*` a po úspešnom vytvorení inštancie triedy `org.hibernate.SessionFactory` získame vytvorené pripojenie do databázy. Parametrom `hibernate.connection.driver_class` definujeme driver pre pripojenie, ktorý používame pre konkrétnu databázu. Pre nás je to `com.mysql.jdbc.Driver`, ktorý sa používa pre *MySQL*. Parametrom `hibernate.dialect` určíme jazyk, ktorý sa má použiť pre dotazy. Nie je nevyhnutné vypĺňať tento parameter, pretože hibernate dokáže väčšinou sám rozpoznať aký jazyk má použiť na základe metadat, ktoré mu vráti príslušný JDBC driver, ktorý vytvorí spojenie. Za povšimnutie stojí takisto parameter

`hibernate.hbm2ddl.auto`, ktorý môže nadobúdať hodnoty `validate, update, create a create-drop`. Pri hodnote `validate`, po vytvorení sedenia validuje správnosť tabuliek v databáze s objektami, vrátane datových typov. Ak obsahuje `create-drop` dokáže vytvoriť podľa objektov príslušné tabuľky a na konci sedenia všetko vytvorené zruší [7].

V konfiguračnom súbore je tiež možné vidieť element `<mapping>`, ktorý odkazuje na mapovací konfiguračný xml dokument konkrétneho objektu. Viac sa o tomto dozviete v kapitole 4.2.2. Vyššie spomínaný kód obsahuje len niekoľko parametrov nastavenia. Zopár ďalších je možné nájsť v [7]. Ak sú všetky atributy dôkladne nastavené, vytvoríme pripojenie pomocou triedy SessionFactory nasledovne:

```
SessionFactory sFactory =
new Configuration().configure().buildSessionFactory();
```

4.2.2 Mapovanie objektov

Objekty, ktoré chceme mapovať do databázy musia byť JavaBean objektami. Nasledujúci kód obsahuje objekt *Mote*, ktorý popisuje uzol senzorovej bezdrôtovej siete:

```
public class Mote {
     private String address;
     private String name;
     private String description;

     public String getDescription() {
            return description;
     }
     public void setDescription(String description) {
            this.description = description;
     }
   public String getName() {
            return name;
     }
     public void setName(String name) {
            this.name = name;
     }
     public String getAddress() {
            return address;
     }
     public void setAddress(String address) {
             this.address = address;
     }
}
```

Každý takýto objekt by mal mať jednoznačný identifikátor, ktorý ho popisuje. Zväčša to býva automaticky inkrementujúci parameter id typu `int` alebo `Long`. V tomto prípade je to parameter `address`, ktorý vyjadruje adresu zariadenia. Tá musí byť v našom systéme jednoznačná. Ak chceme využiť plnú funkčnosť frameworku hibernate, identifikátory sú nevyhnutnou súčasťou.

V kapitole Konfigurácia 4.2.1 sme popísali nastavenia hibernate a pripojení. Jeho súčasťou boli referenčné elementy `<mapping>`, ktoré pri preklade vkladajú konfiguračné súbory. Tie obsahujú informácie o mapovaní jednotlivých objektov. V nasledujúcom kóde je vidieť, ako sú prezentované objekty v databáze. Tento príklad obsahuje mapovanie triedy `Mote` do databázovej tabuľky `mote`:

```
<hibernate-mapping>
     <class name="cz.vutbr.wsagent.entity.Mote" table="mote">
          <id column="address" name="address" type="java.lang.String"></id>
          <property name="name" type="text">
               <column name="name" not-null="false"/>
          </property>
          <property name="description" type="text">
               <column="description" not-null="false"/>
          </property>
     </class>
</hibernate-mapping>
```

Výhodou sú opäť abstraktné definície atribútov, ktoré sa vhodne zvolia pre každú databázu až pri spustení. Element `id` nám definuje primárny kľúč tabuľky. Vložením kódu `<generator class="increment"/>` do jeho tela môžeme spustiť automatické inkrementovanie hodnoty číselnej hodnoty. Okrem tohto generátora existuje množstvo iných s vlastnými metódami inkrementovania. Elementom `property` nastavíme mapovanie jednotlivých atributov triedy s rôznymi parametrami. Takisto môžeme mapovať aj väzby, pre ktoré existujú špeciálne elementy `<many-to-one>` a `<one-to-one>`. Nasledujúci kód popisuje, že trieda Message môže mať väzbu *N ku 1* s triedou Mote:

```
<class name="cz.vutbr.wsagent.entity.Message" table="message">
     <many-to-one cascade="all" class="cz.vutbr.wsagent.entity.Mote"
      name="system">
          <column name="address" not-null="true"/>
     </many-to-one>
</class>
```

Element `<column>` definuje referenciu – cudzí kľúč na kľúč v tabuľke Mote.

Kapitola 5

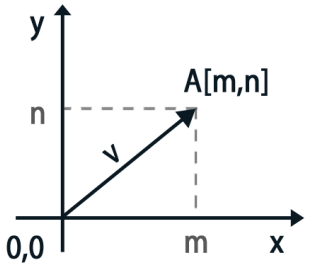
Analytická geometria rovinných útvarov

V tejto kapitole stručne zhrnieme základy analytickej geometrie rovinných útvarov potrebných pre určovanie pozíc objektov topológie siete. Predpokladá sa, že tieto informácie sú čitateľovi známe a len ho uvedú k tejto problematike. Výhradne sa zameriame len na niektoré, pre nás dôležité, útvary. Viac sa dozviete v [17].

5.1 Úvodné definície

Zvolíme **kartézsku súradnicovú sústavu**, pomocou ktorej dokážeme jednoznačne vyjadriť každý geometrický útvar určitou rovnicou. Rovnica potom popisuje výskyt všetkých bodov, ktoré tvoria zvolený útvar. **Bod** je teda v rovine definovaný dvojicou súradníc $[x, y]$. Ľubovoľný bod leží v danom útvare práve vtedy, ak jeho súradnice spĺňajú rovnicu útvaru. Na základe toho možno definovať **prienik útvarov** *U1* a *U2* ako množinu všetkých bodov, ktorých súradnice spĺňajú súčasne rovnice obidvoch útvarov.

Dôležitým geometrickým objektom je **vektor**, ktorý je určený dĺžkou, smerom a rotáciou. Môžeme si ho predstaviť ako orientovanú úsečku, na ktorej je vyznačený začiatočný a koncový bod. Obrázok 5.1 znázorňuje jednoduchý vektor [1].



Obrázek 5.1: Nákres vektora v rovine. [1]

Súradnice vektora zapisujeme ako $v = [v1, v2]$, teda ak je začiatočným bodom bod $A[0, 0]$ a koncovým bod $B[m, n]$, vektor bude mať súradnice $v = [b1 - a1, b2 - a2]$ teda $v = [m, n]$.

Dĺžka vektora je vzdialenosť jeho začiatočného a koncového bodu vyjadrená pomocou

Pytagorovej vety[1]. Značí ho rovnica 5.1.

$$||B - A|| = d(A, B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2} \tag{5.1}$$

5.2 Priamka

Priamku v analytickej geometrií môžeme definovať viacerými spôsobmi [1]:

- **Normálová rovnica**. Ak poznáme jeden bod priamky X_0 a niektorý jej normálový vektor n, potom ľubovoľný bod X leží na priamke vtedy ak vektory $X - X_0$ a n sú navzájom kolmé. Na základe toho dostávame normálovú rovnicu priamky: $(X - X_0) * n = 0$.
- **Všeobecná rovnica**. Získame ju ak rozpíšeme normálovú rovnicu. Jej tvar je definovaný $ax + by + c = 0$, kde a, b a c sú reálne čísla, pričom $n = [a, b]$ a c je rovné skalárnemu súčinu polohového vektora ľubovoľného bodu priamky s normálovým vektorom $[a, b]$.
- **Smernicová rovnica**. Je to rovnica v tvare $y = kx + q$, kde q a k sú reálne čísla. Hodnota k je smernica priamky a je rovná tangensu uhla priamky s kladným smerom osi x. Číslo q je y-ová súradnica priesečníka priamky s osou y.
- **Parametrická rovnica**. Ak poznáme jeden bod priamky X_0 a jej smerový vektor s, tak ľubovoľný bod X leží na danej priamke práve vtedy, ak vektory $X - X_0$ a s sú vzájomne rovnobežné. Potom existuje reálne číslo t (jednoznačne určené bodom), pre ktoré platí: $X - X_0$. Matematicky definované: $x = x_0 + s_1 * t, y = y_0 + s_2 * t$, kde t je z množiny reálnych čísel.

5.3 Kružnica

Kružnica patrí v matematike do skupiny útvarov nazvaných **kuželosečky**. Okrem nej do tejto skupiny patrí **elipsa**, **hyperbola** a **parabola**. Rovnica kružnice so stredom $S = [m, n]$ a polomerom r je znázornená na obrázku 5.2**(a)** a je definovaná rovnicou 5.2.

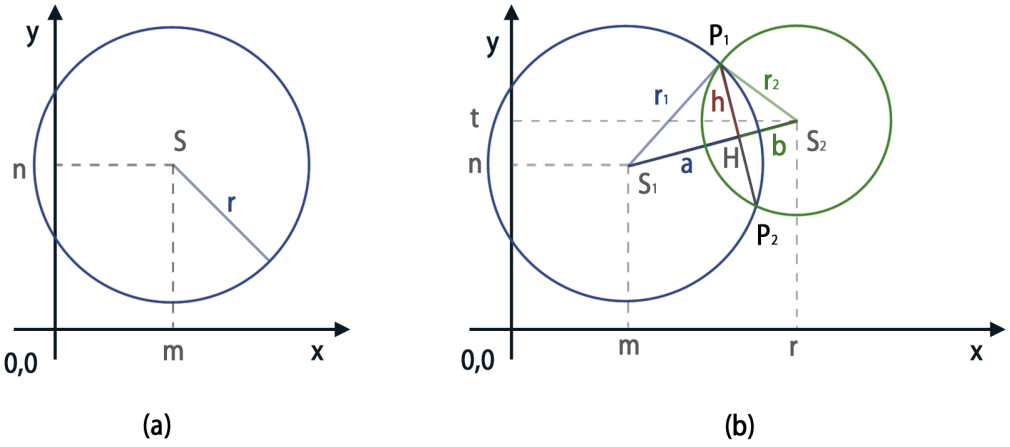
$$(x - m)^2 + (y - n)^2 = r^2 \tag{5.2}$$

Rovnice elipsy, hyperboly a paraboly nie je potrebné pre túto prácu definovať, je ich možné nájsť v [1].

5.3.1 Hľadanie priesečníkov dvoch kružníc

Pri hľadaní priesečníkov dvoch kružníc môžeme prísť k trom druhom výsledkom: žiadny, jeden alebo dva priesečníky. Metóda, ktorú použijeme popisuje obrázok 5.2 **(b)** a má nasledovné kroky [17]:

1. Najprv vypočítame vzdialenosť d medzi stredmi kružníc S_1 a S_2. Ak $d > r_1 + r_2$ neexistuje žiadne riešenie, pretože kružnice sú príliš vzájomne vzdialené vzhľadom k veľkosti ich polomeru. Ak $d < |r_1 - r_2|$ tiež neexistuje riešenie, pretože jedna kružnica sa nachádza v druhej. Ak $d = 0$ a $r_0 = r_1$ tak sú koincidentné a riešení je nekonečne veľa.



Obrázek 5.2: **(a)** Kružnica v rovine. **(b)** Výpočet priesečníkov dvoch kružníc.[17]

2. Pozrime sa na trojuholníky S_1HP_1 a S_2HP_1. Pomocou pytagorovej vety môžeme stanoviť tvrdenie popísané rovnicou 5.3.

$$a^2 + h^2 = {r_1}^2, b^2 + h^2 = {r_2}^2 \tag{5.3}$$

3. Použitím $d = a + b$ dostaneme rovnicu 5.4

$$a = \frac{({r_1}^2 - {r_2}^2 + d^2)}{2d} \tag{5.4}$$

4. Pomocou rovníc 5.3 a 5.4 získame hodnotu pre h: $h = {r_1}^2 - a^2$.

5. Takže bod H môžeme vypočítať rovnicou 5.5.

$$H = S_1 + \frac{a(S_2 - S_1)}{d} \tag{5.5}$$

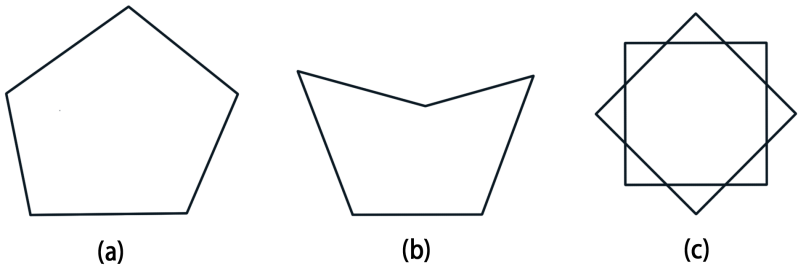
6. A nakoniec rovnice bodov $P_1 = (x_3,\ y_3)$ a $P_2 = (x_4,\ y_4)$ získame vyjadrením 5.6, kde $S_1 = (x_1,\ y_1)$, $S_2 = (x_2,\ y_2)$ a $H = (x_0,\ y_0)$

$$x_{3,4} = x_0 \pm \frac{h(y_2 - y_1)}{d}, y_{3,4} = y_0 \pm \frac{h(x_2 - x_1)}{d} \tag{5.6}$$

5.4 Polygón

Polygón je uzavretý rovinný útvar pozostávajúci z ľubovoľného počtu bodov spojených hranami. Ak sú všetky strany a uhly v ňom ekvivalentné jedná sa o pravidelný polygón. Na základe tvaru ho môžeme rozdeliť na **konvexný**, **konkávny** a **hviezdicový**. Príklad polygónov s týmto rozdelením je znázornený na obrázku 5.3.

Dôležitý poznatok pre nás je výpočet stredového bodu polygónu – **ťažiska**. V konvexných polygónoch sa bude vždy nachádzať vo vnútri, v konkávnych sa môže nachádzať aj mimo jeho plochy.



Obrázek 5.3: Rôzne typy polygónov **(a)** Konvexný. **(b)** Konkávny. **(c)** Hviezdicový. [21]

Ak poznáme všetky body, ktoré polygón tvoria, pozíciu ťažiska môžeme vypočítať rovnicou 5.7.

$$c_x = \frac{x_1 + x_2 + ... + x_k}{k}, c_y = \frac{y_1 + y_2 + ... + y_k}{k} \tag{5.7}$$

V opačnom prípade, ak poznáme len hraničné body pospájané úsečkami použijeme vzorec definovaný rovnicou 5.8 [17].

$$c_x = \frac{1}{6A} * \sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i), c_y = \frac{1}{6A} * \sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \tag{5.8}$$

kde A je obsah polygónu, definovaný pomocou rovnice 5.9 [21].

$$A = \frac{1}{2} * \sum_{i=0}^{n-1} x_i y_{i+1} - x_{i+1} y_i \tag{5.9}$$

Kapitola 6

Návrh architektúry webovej aplikácie

Účelom webového rozhrania je vytvoriť prostredie pre užívateľa systému, ktorý bude môcť pomocou basestation kontrolovať činnosť agentov rozmiestnených v prostredí. Táto kontrola spočíva v monitorovaní prevádzky, ktorá prebieha medzi uzlami. Systém musí umožňovať vkladanie a zaslanie kódu agenta, ktorým je možné aj usmerňovať, s kým sa bude agent dorozumievať na dosiahnutie svojich cieľov. Dôležitým prvkom pri demonštrovaní komunikácie je zobrazenie a spravovanie topológie siete do miery, ktorú systém umožňuje. Systém musí byť navrhnutý vzhľadom na údržbu a pridávanie nových funkcií.

6.1 Popis požiadaviek na webové rozhranie

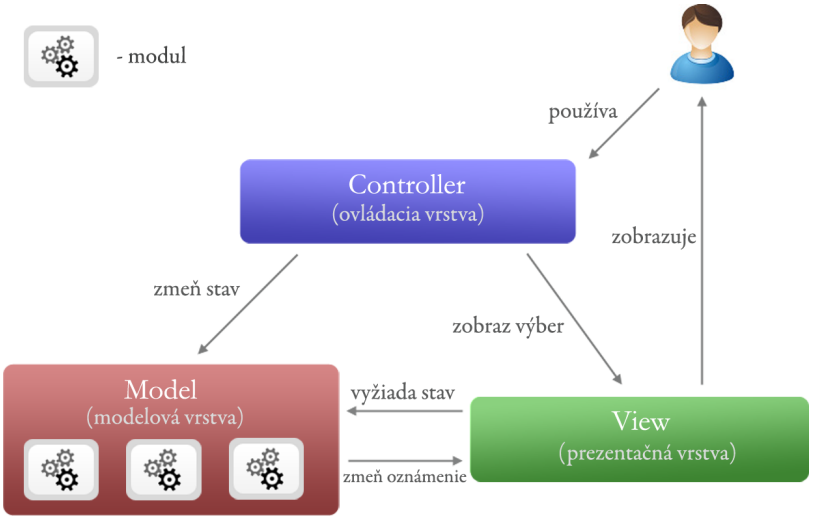
Existujúci systém WSageNt poskytuje rozhranie hlavne pre agentov a správu služieb, ktoré zariadenia ponúkajú. Basestation je spojovacím prvkom medzi nimi a počítačom, pomocou ktorého môžeme tvoriť, upravovať kód agenta a komunikovať s ním. Keďže pre tento účel TinyOS poskytuje knižnice, napísané v Jave, je vhodné ich zúžitkovať. Pre webové rozhrania existuje niekoľko variant technológií, ktoré je možné použiť. V tomto prípade je najvhodnejšie aplikovať technológiu *JavaEE*. Je zároveň kombináciou modelu, ktorý zabezpečuje funkcionalitu, a prezentačnej vrstvy, ktorá zobrazí výsledky.

Rozhranie by malo pozostávať z ovládacích prvkov – modulov, ktoré je možné pridávať, upravovať a odstraňovať, rýchlo a efektívne. Preto je vhodné zaviesť nejakú architektúru, pozostávajúcu z viacerých operačných vrstiev. Možné varianty sú **PCMEF**, **MVC** a **XWA**. Pre navrhovaný systém je vhodná **MVC** (*Model – View – Controller*) architektúra. Skladá sa z *modelovej*, *prezentačnej* a *ovládacej* vrstvy. Obrázok 6.1 zobrazuje nákres klasickej MVC architektúry systému. Pre webové technológie je táto architektúra trochu upravená, kde server vyberá najvhodnejší pohľad na prezentovanie dát.

Webové rozhranie musí poskytovať základné prvky a funkcie informačného systému, ako sú prezeranie rôznych pohľadov, ukladanie, úprava a odstránenie perzistentných objektov. Model, ktorý popisujeme, bude obsahovať objekty agentov, uzlov siete – motes, správ, ktoré sú prenášané pri komunikácií. V systéme by sa ďalej mali vyskytovať základné nastavenia.

Monitorovanie prevádzky siete vyžaduje aktívny a dlhodobý prístup, čo webové technológie veľmi nepodporujú. Tzn. potrebujeme samostatný softvér[1], ktorý by komunikoval

[1] ďalší, okrem webového rozhrania



Obrázek 6.1: Klasická MVC architektúra.

s basestation a spracovával výsledky komunikácie. Spracované dáta by vhodne ukladal do databázy tak, aby ich mohlo webové rozhranie patrične zobraziť.

Keďže služba posielania správ je veľmi energeticky náročná na pamäť, ktorou disponujú mote, nie je reálne možné monitorovať každý paket, ktorý sa pohybuje na sieti. Výhodnejšie je odchytávať celé správy, ktorými medzi sebou mote komunikujú. Okrem správ je v systéme možné posielať aj agentné správy, ktoré takisto môžme zaznamenávať. Tieto údaje môžme neskôr použiť na vyhodnocovanie – vytvorenie rôznych štatistík. Typickým príkladom môže byť vytvorenie modulu pre zobrazenie nameranej teploty na každom uzle v pravidelných časových razítkach. Tieto údaje by sa mohli potom zobrazovať tabulárne alebo pomocou grafov. Takýchto modulov je možné implementovať veľký počet a pripojiť na vytvorené rozhranie.

Stav siete, tzn. ktoré uzly sú aktívne, môžme monitorovať odosielaním prázdnych správ postupne na všetky zaregistrované zariadenia. Každé zariadenie je v systéme identifikované svojou adresou. Tie, ktoré odpovedia, že správu dostali, môžme považovať za prihlásené do siete. Ak vyprší časový interval odosielania správy, uzol nepovažujeme za prihlásený.

Ďalej môžme stav siete monitorovať zobrazovaním topológie, v ktorej budú znázornené všetky uzly poprepájané do grafu. Viac v nasledujúcej kapitole 6.2.

Z hladiska bezpečnosti bezdrôtovej siete neboli vytvorené žiadne opatrenia, preto by sa mohli objaviť útoky (napr. typu *DoS*). Architektúra a možnosti zariadení nie sú však na to vybavené, aby bolo nutné zavádzať mechanizmy, ktoré by boli schopné odolať prípadným útokom.

6.2 Návrh tvorby topológie

Zobrazenie topológie siete nie je triviálny problém. Už pri analýze a návrhu je dôležité správne určiť, akou formou a čo všetko by malo toto zobrazenie obsahovať, v závislosti od potreby systému WSageNt. Aby bezdrôtové siete boli efektívne, používajú sa neobmedzené topológie, bez špecifického usporiadania. Postup tvorby je sieťový (viď kapitola 2.5), čo znamená, že sa v sieti nachádza prepojovací uzol – basestation.

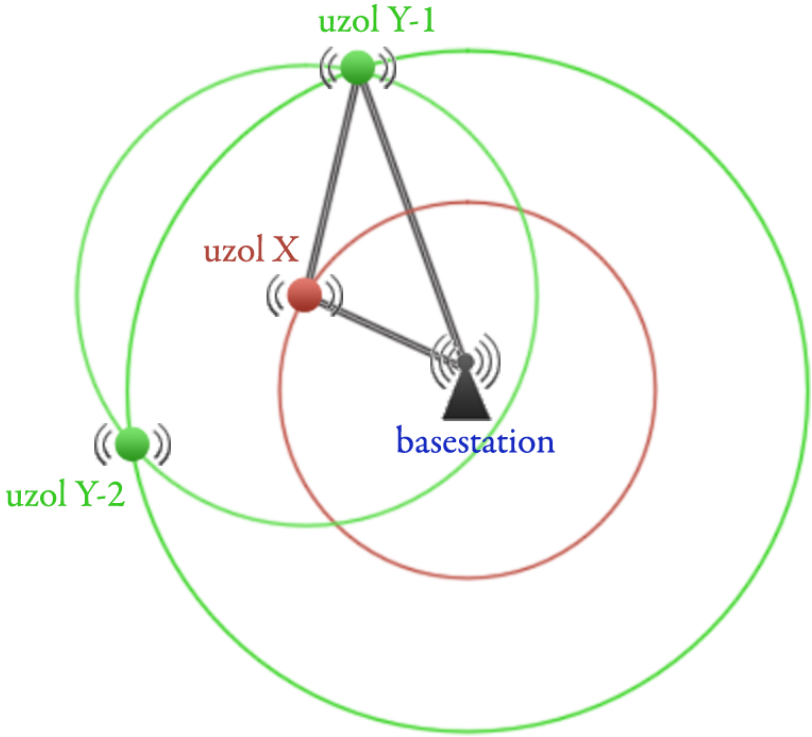
K podrobnejšej znalosti o sieti sa môžeme dostať aplikovaním lokalizačných metód.

Problémom je, že v tomto systéme má známu pozíciu len basestation, čo je málo na úplné určenie polohy uzla v sieti. Možnosťou je vytvoriť približnú lokalizáciu, založenú na aproximáciách. Tieto aproximácie môžu byť založené na postupnom tvorení grafu topológie. Pre tento systém je toto zobrazenie postačujúce, nie je dôležité presne určiť výskyt uzlov v súradnicovej osi.

Použitím RSSI metódy získame vzdialenosť objektov. Tzn. môžeme mať k dispozícií informácie o jednotlivých vzdialenostiach uzlov. Tieto informácie sa v rámci dvoch uzlov však môžu odlišovať. Napríklad akýkoľvek uzol A prijme signál od uzlu B so silou signálu 62%. Uzol B odpovie a signál dorazí k A so silou 54%. Tieto odlišné údaje síl signálov, spôsobené fyzikálnymi fenoménmi (viď kapitola 2.3.1), zapríčinia odlišné výstupy RSSI algoritmu (viď kapitola 3.4), tým aj divergentné vzdialenosti. Snáď jediným riešením je použiť vážený priemer, pretože skutočná sila signálu je pravdepodobne v intervale týchto dvoch hodnôt, v prípade, že sú obe mote kalibrované rovnako.

Pri tvorení grafu je možné tieto vzdialenosti využiť a pomocou analytickej geometrie približne nájsť uzly siete. Čím viac uzlov sa bude nachádzať v sieti o to presnejšia môže byť aproximácia jednotlivých uzlov. Z výsledkov experimentálneho merania sily signálov vieme, že im zodpovedajúce vzdialenosti budú veľmi neurčité, preto je potrebné ich do najvyššej miery prispôsobovať pri hľadaní grafu.

Obrázok 6.2 ukazuje prípad, kedy sú v systéme s basestation len 2 uzly X a Y. Pomocou rovníc analytickej geometrie sme určili vzájomnú polohu. Vznikli dve rôzne možnosti *Y-1* a *Y-2* výskytu uzlu Y vzhľadom na X. X sa môže nachádzať kdekoľvek na červenej kružnici, Y na zelenej. Ich vzájomná poloha bude vždy rovnaká. Každým nový uzol, ktorý začleníme do sústavy, môže spôsobiť vznik novej varianty vzhľadu grafu a tiež môže dopomôcť k spresneniu tých existujúcich.



Obrázek 6.2: Náčrt tvorby topológie s aproximovanou lokalizáciou

V tomto prípade sa môžeme rozhodnúť pre ľubovoľný z dvojice bodov *Y-1* a *Y-2*, pretože určovanie nasledovných uzlov bude tvoriť vzájomne symetrický graf oboch možností. Ak by však bolo určovanie postavené len na tejto metóde, mohlo by sa stať, že by sme hľadali prienik kružníc k bodom vzájomne veľmi vzdialeným, pričom v skutočnosti by sa nachádzali blízko pri sebe.

Spravovanie topológie môžeme tiež do určitej miery zaviesť do systému uspaním niektorých uzlov na určitú dobu, hlavne v husto rozmiestnenej sieti. Systém smerovanie nepodporuje kvôli nízkej pamäti DRAM, ktorou disponuje mote, takže komunikácia môže prebiehať len priamo – od odosielateľa k prijímaču.

6.3 Analýza a návrh softvéru pomocou UML

V tejto fáze vývoja je vhodné vytvoriť model, ktorý by nám popisoval vytváranú aplikáciu. **UML** (*Unified Modeling Language*) nám to umožní. Výhodou je, že nie je viazaný na žiadnu špecifickú metodiku alebo životný cyklus. Poskytuje nám len vizuálnu syntax, na základe ktorej môžeme vyvíjanú aplikáciu jednoducho implementovať. Zámerom jazyka UML spolu s metodikou **UP** (*Unified Process*), od ich vzniku, bola podpora najlepších postupov používaných v softvérovom inžinierstve vychádzajúcich z overených skúseností. K tomuto účelu boli v týchto jazykoch unifikované všetky predchádzajúce pokusy o tvorbu jazykov pre vizuálne modelovanie a proces softvérového inžinierstva. [10] V tejto podkapitole sa nebudem podrobne venovať jazyku UML, len načrtnem základné diagramy vytvorené pre modelovanie webového rozhrania.

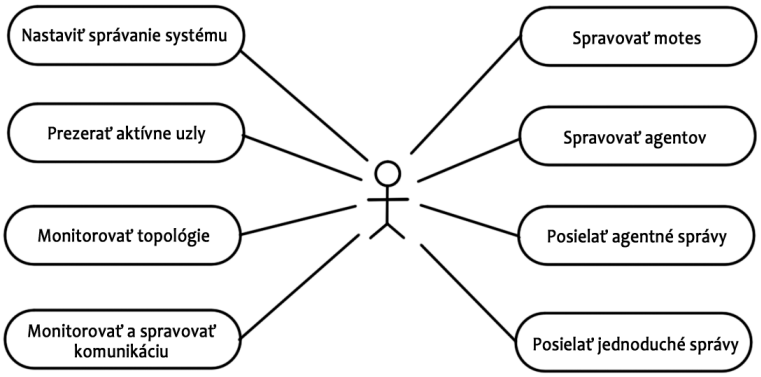
6.3.1 Model prípadov užitia

Modelovanie prípadov užitia je jednou z foriem získavania požiadaviek. Jeho výstup je diagram, ktorý obsahuje štyri komponenty: *účastníci*, *prípady užitia*, *relácie a hranice* systému. [10]

Navrhovaný systém bude obsahovať len jednu rolu účastníkov. Bude to užívateľ, ktorý bude môcť manipulovať so všetkými funkciami - prípadmi užitia. Od toho sa odvíjajú aj hranice systému. Predpokladá sa však, že vyvíjaný systém bude môcť byť rozšíriteľný, čím sa jeho hranice zmenia. Obrázok 6.3 obsahuje diagram prípadov užitia systému. Je zreteľné, že užívateľ bude mať niekoľko základných možností. Jednou, veľmi dôležitou, z nich je posielanie správ. Špecifikáciu prípadu užitia poslania agentnej správy nájdete v prílohe 10.1. Agentné správy môže tiež užívateľ spravovať, tzn. okrem poslania ich môže uložiť, aby ich mohol použiť aj neskôr. Okrem toho musí systém podporovať spravovanie uzlov siete – motes. V prípade, že zaevidujeme príchod novej správy od nejakého uzlu, systém by ju mal zobraziť. Takisto môžme sledovať aj správy, ktoré nie sú určené pre basestation. Ďalšou vlastnosťou je zobrazenie topológie siete. Užívateľ by mal byť schopný určiť akým spôsobom chce získavať údaje pre toto zobrazenie. V neposlednom rade by mal mať možnosť spravovať základné nastavenia systému.

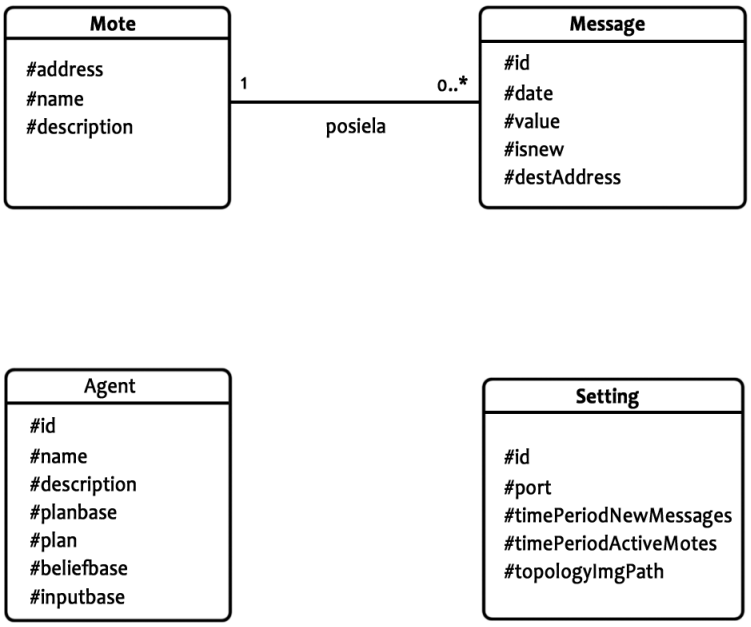
6.3.2 Diagram tried

V predchádzajúcej kapitole sme vytvorili model prípadov užitia, ktorý je dôležitý pre modelovanie tried. Diagram vytvoríme len z tried a väzieb, ktoré popisujú modelované prostredie a objekty, ktoré v ňom vystupujú.



Obrázek 6.3: Diagram prípadov užitia pre navrhovaný systém.

Obrázok 6.4 zobrazuje návrh tried modelovaného systému. Hlavným objektom je trieda `Mote`, ktorá popisuje zariadenie – uzol siete. Je definovaná adresou a môže obsahovať názov a popis konkrétneho zariadenia. Ďalším objektom je správa (`Message`), ktorú uzol posiela inému uzlu. Okrem vlastného obsahu zaznamenáva dátum a čas príchodu, stav (nová, prečítaná) a cieľová adresa zariadenia, ktorému bola poslaná. Naopak zdrojová adresa je definovaná cudzím kľúčom zariadenia, ktoré ju odoslalo. Ďalšími objektami je `Setting`, ktorý obsahuje nastavenia, a `Agent` popisujúci správanie agenta.



Obrázek 6.4: Diagram tried pre navrhovaný systém.

Kapitola 7

Implementácia systému

V nasledujúcej kapitole popíšeme systém, ktorý bol implementovaný. Rozoberieme prvky, vrstvy a moduly, z ktorých sa skladá. Stručne vysvetlíme základné princípy použité pri jeho tvorbe a funkcie, ktoré poskytuje.

7.1 Základné prvky systému

Implementovaný systém sa skladá z dvoch samostatných, spustiteľných aplikácií:

- konzolová aplikácia **BSComm**, ktorá slúži na komunikáciu s basestation.
- webové rozhranie **ControlPanel**, ktoré zobrazuje výsledky monitorovania a je nástrojom pre ovládanie aktívnych mote.

Obidve aplikácie boli vyvíjané v súlade s fázou návrhu. Ako základ bola použitá platforma *Java* a vývojové prostredie *NetBeans IDE 6.7.1*. Pre účely uchovania dát bol použitý framework *Hibernate*. Stavba systémov bola rozložená vo vrstvách do balíčkov, podľa konkrétnej architektúry. Všetky balíky patria pod jeden kmeňový s názvom `cz.vutbr.wsagent`. Väzby jednotlivých balíkov sú smerom od vyššej vrstvy k tej nižšej. V tejto štruktúre sa nachádza aj balík `cz.vutbr.wsagent.entity`, ktorý obsahuje triedy popisujúce modelované prostredie a jeho objekty.

7.1.1 BSComm

Táto konzolová aplikácia bola vytvorená pre účely komunikácie s basestation. Jej úlohou je preposielanie správ od webového rozhrania do basestation a naopak. Bola vyvinutá na platforme **Java SE**.

Riadiacou zložkou v programe je balík `cz.vutbr.wsagent.bscomm`, ktorého trieda `Main` obsahuje spúšťací mechanizmus. Na vstupe očakáva dva parametre:

- Cesta k sériovému portu, kde je pripojená basestation (napr. *serial@/dev/ttyUSB1:iris*).
- Port k soketu, na ktorom prebieha komunikácia s webovým rozhraním (napr. *7777*).

Ak sú vstupné parametre v poriadku, založí sa spojenie s basestation, vytvorením inštancie triedy `Comm` v balíku `cz.vutbr.wsagent.comm` (Viac v sekcií 7.1.1). Pomocou triedy `BSServerSocket` sa otvorí soket na danom porte, na ktorom sa čaká na požiadavky. Nasleduje hlavná riadiaca slučka, ktorá je v činnosti, kým nie je program ukončený ukončujúcim signálom. Obsluha požiadaviek začína metódou `Listen()` triedy `BSServerSocket`,

ktorá pasívne čaká na sokete a pri príchode získa obsah požiadavky, ktorý vráti do hlavnej riadiacej slučky. Po zistení, či sa jedná o agentnú správu alebo jednoduchú správu, vytvorí prepravku[1] triedy `Messenger`, do ktorej vloží adresu prijímateľa a objekt odosielanej správy. Prostredníctvom inštancie triedy *Comm* vykoná odoslanie. Následne je vytvorený poslucháč[2] triedy `SendingResultListener`, ktorý čaká určitý časový interval na výsledok odosielania správy. V prípade, že je prekročená povolená časová doba, inicializuje výnimku `TimeExpiredException`. Výsledok komunikácie odošle späť logike webového rozhrania pomocou metódy `sendResponse()`. Tým prechádza na začiatok slučky a cyklus sa opakuje.

Komunikácia s webovým rozhraním je definovaná protokolom, ktorý popisuje trieda `InternalCommProtocol` v balíku `bscomm`. V ňom sa nachádza aj nástroj (`MessageParser`), ktorý syntakticky rozoberie prijatú správu v súlade s protokolom.

Trieda Comm

Trieda Comm spĺňa 3 základné funkcie komunikácie:

- Vytvorenie stáleho, neprerušovaného spojenia.
- Odoslanie jednoduchej alebo agentnej správy.
- Príjem správy a oboznámenie poslucháčov, že správa prišla.

Aby mohla trieda Comm patrične reagovať na prichádzajúce správy, musí implementovať triedu `MessageListener`, balíku `net.tinyos.message`. Tento balík spolu s inými z knižnice `tinyos` zabezpečuje stabilnú komunikáciu na nižšej vrstve. Našou úlohou je teda len deklarovať, čo s dátami, ktoré z basestation prídu. To vykonáme pomocou metódy `messageReceived()`, ktorú, ako jedinú musíme implementovať.

Ešte pred tým sa pozrime na vytvorenie spojenia. Zabezpečuje ju trieda `PhoenixSource` a `moteIF`. Prvá menovaná aplikuje komunikačné funkcie na transportnej úrovni. Jej inštancia poskytuje vlákna, ktoré pracujú s konkrétnymi paketmi. `MoteIF` používa `PhoenixSource`, aby mohla poskytovať spomínanú aplikačnú úroveň. Ak chceme prijímať správy prostredníctvom metódy `messageReceived()`, musíme vo svojej inštancí tejto triedy zaregistrovať poslucháčov. Nasledujúci kód zaisťuje spomínané potreby:

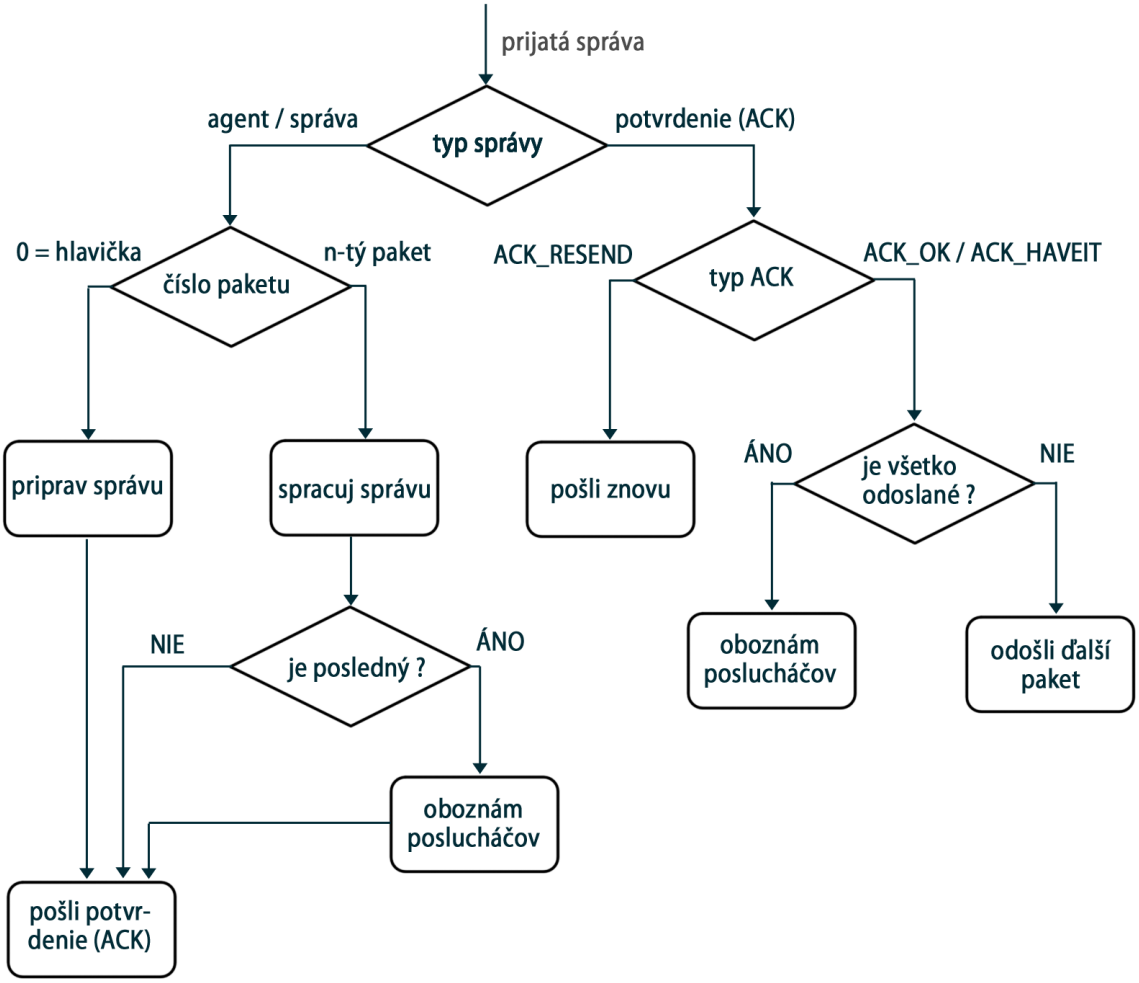
```
// priklad: port = serial@/dev/ttyUSB1:micaz
phoenix = BuildSource.makePhoenix(port, PrintStreamMessenger.err);

// inicializácia komunikačného modulu
moteIF = new MoteIF(phoenix);
// prihlásenie sa k odberu
moteIF.registerListener(new RadioClassAgent(), this);
moteIF.registerListener(new RadioClassMessage(), this);
moteIF.registerListener(new RadioClassAck(), this);
```

Kód obsahuje zaregistrovanie troch poslucháčov: pre príjem agentnej a jednoduchej správy a potvrdzujúceho paketu.

Dôležitou vlastnosťou triedy `Comm` je, že pre príjem aj každé odoslanie sa používa tá istá inštancia. Z toho dôvodu sa musia na niektorých miestach používať monitory na jednotlivý

[1] podľa návrhového vzoru Crate
[2] podľa návrhového vzoru Observer od skupiny GoF

prístup k atribútom a pod. Ďalšou črtou je posielanie a príjem správ po paketoch s potvrdzovaním výsledku, čo sťažuje uchovávanie dát, ktoré ešte neboli poslané alebo prijaté. Preto boli zavedené privátne objekty `sendPacketHashMap` a `recPacketHashMap` typu `HashMap`. Sú to zoznamy, ku ktorým sa pristupuje identifikátorom, v tomto prípade adresou zariadenia. V zásade platí, že ak chceme poslať správu do basestation, pošleme paket s číslom 0 a zvyšok neodoslanej správy uložíme do mapy `sendPacketHashMap`, s identifikátorom adresy správy kam sa odosiela. Analogicky to platí aj pre príjem. Obsah správy modeluje trieda `TransPacket`.



Obrázek 7.1: Stavový diagram príjmu správy.

Odosielanie agentných a jednoduchých správ poskytujú verejné metódy `sendAgent()` a `sendMsg`, ktoré spracujú správu a vytvoria hlavičku. Ešte pred tým skontrolujú, či medzi cieľovou adresou a basestation neprebieha komunikácia (neboli všetky pakety odoslané alebo prijaté). Ak áno vyhodí vytvorenú výnimku `DeviceBusyException`. V opačnom prípade pošlú pomocou rozhrania hlavičku do basestation na uzol s požadovanou adresou. Telo správy sa uloží do objektu `sendPacketHashMap`, ktorá bude postupne odoslaná v nasledujúcich paketoch, v okamihu keď od uzla príde odpoveď o úspešnom prijatí. Tento cyklus dohliada na konzistentnosť komunikácie pomocou niekoľkých kontrolných prvkov. Sú to

kontrola poradia paketov, dĺžky zatiaľ neodoslanej správy a kontrolného súčtu.

Prijímanie správ funguje na podobnom princípe. Zabezpečujú to zaregistrovaný poslucháči inštancie triedy `MoteIF`. Nové vlákno, ktoré sa vytvorí pomocou inštancie triedy `PhoenixSource`, zaobstará volanie metódy `messageReceived`, ktorá sa postará o spracovanie dát. Tým začína proces, ktorý popisuje obrázok 7.1.

Na úvod zistíme o aký typ správy sa jedná. Môže to byť agentná alebo jednoduchá správa alebo potvrdenie. Prvé dve možnosti riešime rovnako, pretože v nich nie je v zásade žiadny rozdiel. Agentná správa obsahuje agentný kód, ktorý popisuje jeho správanie atď. Jednoduchá správa väčšinou uchováva informácie z merania senzorov. V tomto momente musíme zistiť, ktorý paket prišiel. Ak je to 0-tý, jedná sa o hlavičku. V druhom prípade z paketov postupne získavame obsah prichádzajúcej správy. Rovnako ako aj v odosielaní prebieha kontrola integrity dát. V každom prípade sa odosiela naspäť výsledok doručenia. Ak získame posledný paket oboznámime poslucháčov rozhrania `IReceivingListener` o prijatej správe[3].

Ak bol v prvom kroku typ správy potvrdenie, znamená to, že sme v určitom čase pred tým uskutočnili odosielanie, takže niektorý uzol nám oznámil jej výsledok. Následne musíme zistiť, o aký druh potvrdenia sa jedná. Ak je to `ACK_RESEND`, požaduje o opätovné zaslanie celej správy z dôvodu porušenia. Ak je to `ACK_OK` alebo `ACK_HAVEIT` znamená to, že paket dorazil v poriadku, alebo ho už v minulosti prijal. Pri odosielaní ďalšieho paketu musíme zistiť, či je ešte čo odosielať. Ak nie, oboznámime poslucháčov (*triedy* `ISendListener`), že správa bola kompletne a úspešne poslaná. V opačnom prípade posielame v poradí nasledujúci paket.

7.1.2 ControlPanel

Okrem samotných JAVA EE technológií bol veľkou oporou tohto webového rozhrania framework Struts 2. Pomocou neho bola architektúra aplikácie rozdelená do vrstiev. Tie sa ešte rozdelili do podvrstiev (balíčkov a zložiek) podľa logickej stavby nasledovne:

- **Vrstva modelu** obsahuje aplikačnú logiku a prácu s modelom. Rozdelená do balíkov:
  - `client` pre komunikáciu s basestation pomocou BSComm.
  - `entity` obsahuje objekty modelu (väčšinou JavaBean objekty).
  - `util` slúži na podporu roznordých mechanizmov. Obsahuje triedy na spracovanie prijatých správ, síl signálov a vygenerovanie obrázkov grafov.
  - `db.hibernate` modeluje perzistentné uchovanie objektov pomocou hibernate frameworku.
- **Správa akcií** pozostáva z:
  - Xml konfiguračných súborov, ktoré sa nachádzajú v balíku `<default-package>`.
  - balíkov `controlpanel` a `controlpanel.system`, ktoré deklarujú logiku akcií.
  - balíka `controlpanel.exception`, ktorý obsahuje špecifické výnimky, ktoré môžu vzniknúť pri práci s akciami.

[3] V tejto verzí to bol iba posluchač triedy `ReceivingListener` z balíku `BSComm`, ktorý každú prichádzajúcu správu uložil do databázy.

- **Prezentačná vrstva**. Ďalej bola rozdelená do adresárovej štruktúry:
  - *ground* – Obsahuje kostru grafického užívateľského rozhrania, vrátane štýlov a javascript funkcií.
  - *system* – Obsahuje základne JSP stránky systému (nastavenia, o aplikácií a pod.).
  - *wsagent* – Zahŕňa JSP stránky pre zobrazenie dát a prácu so systémom WSageNt.
  - *topologyImg* – Predvolená zložka určená k ukladaniu obrázkov topológii.

Vrstva modelu

Abecedne prvý logický prvok modelu, balíček `client`, sa stará o vnútornú komunikáciu s aplikáciou BSComm. Zabezpečuje ju najmä trieda `ClientSocket`, ktorej vytvorenie inštancie zaistí pripojenie s aplikáciou BSComm. Tá je spustená na lokálnom počítači a definuje ju port, na ktorom beží. Metóda `send()` sa postará o poslanie správy a následne čaká na výsledok komunikácie, daný hodnotou podľa protokolu (viď kapitola 7.2). Ak je výsledok úspešný, skončí a odovzdá riadenie. Ak nie vyhodí takú výnimku, ktorá charakterizuje neúspech. Komunikácia prebieha pod protokolom, ktorý je definovaný triedou `InternalCommProtocol`. Okrem nej sa v balíku nachádza aj trieda `MessageConstructor`, ktorá vytvorí správu pre odoslanie.

Balíček s názvom `entity` sú objekty, ktoré popisujú model – jeho vlastnosti a správanie. Okrem toho zahŕňa pre jednotlivé objekty aj rozhrania, ktoré definujú prácu s databázou.

Balík `util`, tak ako v každej inej knižnici, predstavuje podporu rôznych druhov výpočtových a iných algoritmov. V tomto systéme bol použitý na aplikovanie monitorovania. Nachádzajú sa tu nástroje pre počítanie so silou signálu, matematické rozhranie analytickej geometrie, ktoré slúži pre vytvorenie grafu topológie a nástroje pre jej vykreslenie.

Správa akcií

Pridelenie a definovanie akcií sa nachádza v xml konfiguračných súboroch, ktoré sú umiestnené v základnom balíku (`<default-package>`). Hlavným z nich, do ktorého sú pri preklade umiestnené všetky ostatné, je *struts.xml*. Ďalšími sú `system.xml`, pre definovanie základných akcií systému a `wsagent.xml`, ktorý popisuje špecifické akcie pre moduly.

Balíky `controlpanel` a `controlpanel.system` obsahujú aplikačnú logiku akcií. Hlavnou triedou, ktorá sa tu nachádza, je `ControlPanelAction`. Obsahujú atributy, ktoré majú byť viditeľné v každej triede, preto ju aj rozširujú. Aby sme mohli využiť všetky vlastnosti, ktoré nám Struts 2 ponúka, `ControlPanelAction` rozširuje základnú triedu pre akcie `ActionSupport` (viď kapitola 4.1.1).

Pri práci s akciami bolo definovaných niekoľko výnimiek, ktoré môžu vznikať prevažne pri komunikácií s BSComm, s databázou a pod. Všetky sa nachádzajú v triede `controlpanel.exception`.

Prezentačná vrstva

Prezentačná vrstva bola implementovaná podľa návrhu grafického užívateľského rozhrania. Základným stavebným kameňom sú JSP stránky, ktoré predstavujú jednotlivé pohľady. Základnou komponentou je zložka `ground`, v ktorej sa nachádzajú iné grafické komponenty – moduly. Základom každého pohľadu je GUI kostra, ktorá ho zjednocuje pre celý systém. Nasledujúci kód demonštruje použitie kostry s elementami frameworku Struts 2:

```
<body>
    <s:div id="content" >
        <s:include value="/ground/Header.jsp" />
        <s:include value="/ground/Menu.jsp" />
        <s:include value="/ground/Right.jsp" />

        <!-- The core of every page -->
        <s:div id="main">
            <h2>Template page</h2>
            This is the template page.
        </s:div>

        <s:include value="/ground/Footer.jsp" />
    </s:div>
</body>
```

Pri preklade JSP stránky do HTML sú jednotlivé moduly vložené do dokumentu, ktorý ich vyžaduje.

Aktívny panel

Aktívny panel, ako už názov napovedá, je GUI komponenta, ktorá dynamicky mení svoj obsah, bez toho aby si ju užívateľ sám vyžiadal. Tento prvok slúži na to, aby upozornil užívateľa na zmeny a udalosti v systéme. Aktuálne aktívny panel obsahuje dve funkcie:

- **Kontrola nových, prijatých správ**. Ak príde nová správa, zmení sa signalizačná ikona obsahujúca písmeno N[4]. Je možné na ňu kliknúť, po čom sa zobrazí stránka so všetkými (aj novou) správami uzla, ktoré nám ju poslalo.
- **Monitorovanie pripojených uzlov**. Ak sa určitý uzol prevedie do aktívneho stavu[5], alebo sa priblíži na vzdialenosť vzájomného dosahu, je zobrazený v zozname *Online*.

Keďže požadujeme, aby sa nachádzal tento panel na každej stránke systému, musí byť vložený do každého dokumentu. Jeho implementácia sa nachádza v súbore `Right.jsp`. Náhľad aktívneho panela je na obrázku 7.2. Je vidieť, že v čase vytvorenia tohto obrázku boli pripojené dva uzly a žiadna nová správa nebola prijatá.

Obrázek 7.2: Náhľad aktívneho panela.

[4] N ako New alebo Nová
[5] je zapnutý a schopný prijímať správy

Tieto funkcie sú zabezpečené pomocou technológie **AJAX** (*Asynchronous JavaScript and XML*). Pre Struts 2 existuje framework `dojo`, ktorý nám ponúka niekoľko objektov, podporujúcich asynchronnú komunikáciu so serverom.

Pre prácu s objektami umiestnime do JSP stránky značku pre použitie knižnice:

```
<%@ taglib prefix="sx" uri="/struts-dojo-tags" %>
```

a do hlavičky každej stránky kód, ktorý pri preklade vloží potrebné `AJAX` funkcie:

```
<sx:head/>
```

Výhodou je, že podporované objekty priamo pracujú s akciami, takže ich netreba k nim nijako prispôsobovať. Nasledujúci kód prezentuje volanie akcie `CheckOnlineMotes` pre získanie dostupných uzlov siete. Tento proces prebieha v určitých časových intervaloch, ktorých hodnota je získaná pomocou OGNL výrazu z atributu `timePeriodActiveMotes` inštancie triedy `Setting`.

```
<s:url id="action2" action="CheckOnlineMotes"/>

<sx:div id="checkOnlineMotes" href="%{action2}" listenTopics="/refresh"
    showLoadingText="false" updateFreq="%{setting.timePeriodActiveMotes}"
    startTimerListenTopics="/startTimer"
    stopTimerListenTopics="/stopTimer" >
</sx:div>
```

Výsledkom volanej akcie je zobrazenie dokumentu `ActiveMotes.jsp`, ktorá obsahuje zoznam aktívnych uzlov.

7.2 Komunikačný protokol

Komunikačný protokol slúži na definovanie komunikácie medzi BSComm systémom a webovým rozhraním Control Panel. Priebeh komunikácie je znázornený na obrázku 7.3.

ControlPanel posiela požiadavky so správami, ktoré majú byť odoslané určitému uzlu siete pomocou BSComm pripojenému na basestation. BSComm ich spracuje a odošle pomocou rozhrania do basestation, ktoré ich pošle na požadovaný uzol. Vytvorená požiadavka v sebe obsahuje správu a adresu koncového uzla. Správa môže byť agentná alebo jednoduchá (obsahujúca dáta a pod.), aj keď použitý protokol poskytuje použitie akéhokoľvek typu.

Formát novo-vytvorenej agentnej správy je nasledovný:

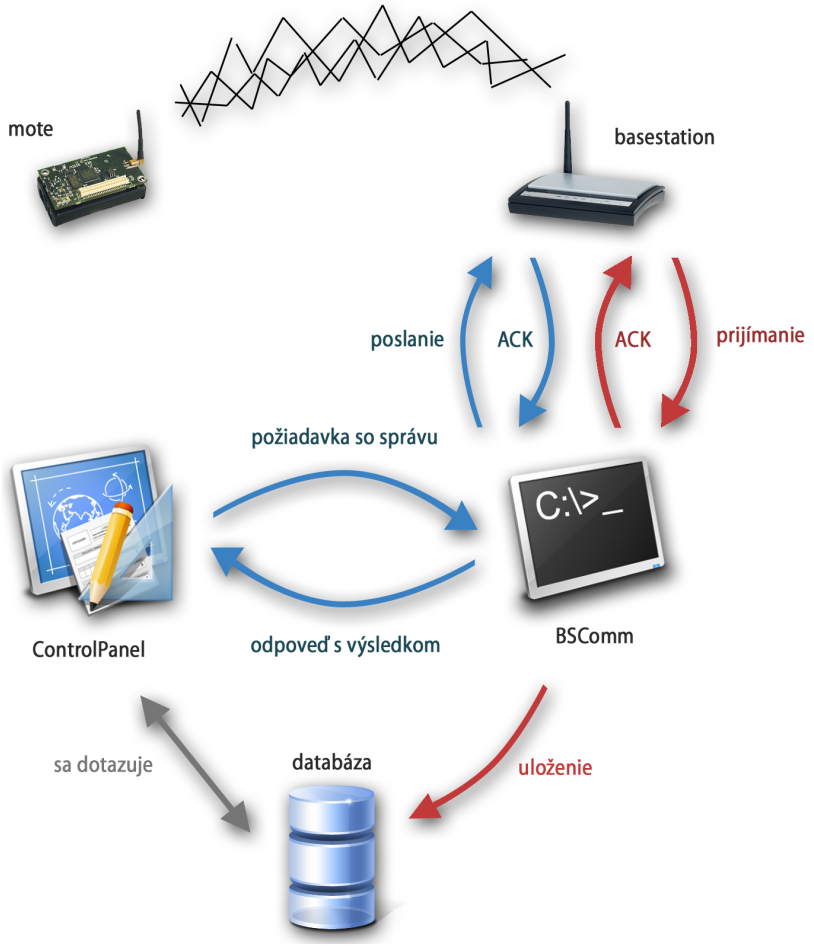
```
typ_správy ODDEĽOVAČ adresa_uzla ODDEĽOVAČ planBase ODDEĽOVAČ
                    plan ODDEĽOVAČ beliefBase ODDEĽOVAČ inputBase
```

Formát jednoduchej správy sa od agentnej veľmi nelíši:

```
typ_správy ODDEĽOVAČ adresa_uzla ODDEĽOVAČ obsah_správy
```

Príkladom môže byť:

```
M;.,WS,-;3;.,WS,-;(100)
```



Obrázek 7.3: Priebeh komunikácie.

Pri analýze som dospel k záveru, že nie je nevyhnutné komunikáciu šifrovať kvôli bezpečnosti a tiež ju nie je dôležité kódovať vzhľadom na prenášaný obsah. Tým zbytočne nezdržujeme a nezaťažujeme priebeh komunikácie. Ak by aktuálny oddeľovač nevyhovoval iným jazykom, ktoré by sme chceli použili v obsahu správ, jednoducho zvolíme iný.

Odpoveď je číselná hodnota, ktorá značí úspech odosielania na konkrétny uzol. Aktuálne implementované odpovede je možné nájsť v tabuľke 7.2.

Číslo	Konštanta	Význam
0	`SENDING_SUCCES`	Úspech operácie
1	`SENDING_ERROR`	Neurčitá chyba
2	`SENDING_EXPIRED`	Čas odosielania vypršal
3	`SENDING_DEV_BUSY`	Rozhranie už komunikuje so zariadením

Tabulka 7.1: Aktuálne implementované odpovede v súlade s protokolom.

Ak BSComm aplikácia prijme ľubovoľnú správu (ňou nemyslíme potvrdenie), uloží ju do databázy s poznámkou, že je voľná. Rozhranie ControlPanel si ju potom vyzdvihne.

Všetky konštanty, ktoré sú použité v protokole sa nachádzajú v rozhraní s názvom

`InternalCommProtocol` v oboch aplikáciách a objekty, ktoré s týmto rozhraním priamo pracujú, ho implementujú.

7.3 Implementácia databázy

Databáza bola implementovaná vo frameworku Hibernate v súlade s diagramom tried, popisujúceho entity modelu. Bolo vytvorených niekoľko konfiguračných súborov, pričom každý z nich definuje mapovanie určitej entity. Pomocou továrenskej triedy `DbConnFactory` je možné získať otvorené pripojenie do databázy. Toto pripojenie využívajú objekty s príponou `DAO` (*Data Access Object*), ktoré implementujú rozhrania[6] v balíku entity. Okrem toho sa tu nachádza továrenská metóda `DAOFactory`, ktorá poskytuje inštancie k jednotlivým DAO objektom. Ak teda chceme niečo vložiť do databázy, napr. mote, použijeme konštrukciu:

```
DAOFactory.getDAOFactory().getMoteDAO().save(moteBean);
```

Parameter moteBean je inštancia objektu, o ktorého perzistentné uloženie sa postará framework hibernate v ľubovoľnom databázovom softvéri, ktorý podporuje. Vyberanie, editácia a mazanie objektov z databázy prebieha na rovnakom princípe.

[6] podľa návrhového vzoru Adaptér

Kapitola 8

Implementácia topológie

V nasledujúcej kapitole sa budeme zaoberať riešeniami a postupmi tvorby topológie a jej výslednou implementáciou. V úvode rozoberiem získavanie údajov síl signálov RSSI, pomocou ktorých sa určuje graf. Ďalej sa budeme zaoberať určovaním polôh na základe týchto hodnôt. Rozoberieme základné problémy, ktoré pri tom môžu vzniknúť. Na záver popíšeme základné prvky a princípy použité pri implementovaní vymedzovania a vykreslenia topológie.

8.1 Dopyt a spracovanie RSSI

Ak sú dve zariadenia aktívne a môžu so sebou komunikovať, s určitým výkonom vysielajú signál, ktorý vo všeobecnosti slabne pri čoraz väčšej vzájomnej vzdialenosti. Ak vytvoríme bezdrôtovú sieť z takýchto zariadení, musíme počítať s tým, že v sieti sa budú nachádzať uzly, ktorých signál nebude dostatočne silný na to, aby spolu mohli komunikovať. Pri tvorbe topológie potom vyžadujeme, aby sme mali informáciu o každom uzle siete. V systéme WSageNt si to vyžaduje agenta, ktorý by bol schopný sa presúvať z mote na mote a zisťovať sily signálov okolia. Výsledné údaje by poslal na basestation, ak aj v iných prípadoch, pomocou agentnej mobility. Kvôli parametrom, ktorými disponujú mote to nie je triviálny problém, ktorý momentálna architektúra celkom nerieši.

Pri získavaní a spracovaní sily signálu postupujeme v podstate rovnako, ako pri posielaní ľubovoľnej agentnej správy na určité zariadenie. Aby sme získali čo najväčší okruh prostredia, pošleme túto správu na všetky aktívne zariadenia. Tie nám odpovedajú opäť agentnou správou s požadovanými informáciami. V zásade je postup nasledovný:

- Pomocou ALLL kódu (viď kapitola 3.3) zostavíme príkaz na získanie sily signálu okolia mote, ktorému tento príkaz pošleme. Príkaz môže byť nasledovný: `$(n)$(m,(1))`.
- Ak nám mote odpovedal neprázdnou správou s očakávaným obsahom, získame ju z databázy, kde bude označená ako nová.
- Spracujeme správu tak, aby sme získali jednotlivé dáta vo formáte (*zdroj*, *cieľ*, *sila signálu*), kde zdroj je adresa zariadenia, ktoré nám správu poslalo a cieľ je adresa mote, ktorý príjma signál o danej sile.

Získali sme zoznam vzájomných dosahov jednotlivých uzlov siete. Vo väčšine prípadov sa budú záznamy opakovať, pretože každý z dvojice komunikujúcich uzlov nám posiela namerané hodnoty. Vtedy je rozumným riešením vypočítať aritmetický priemer týchto dvoch

hodnôt síl signálu a uchovávať len jeden záznam s danou hodnotou. Výnimka môže nastať pri veľkej vzdialenosti, kedy len jeden z dvojice dokáže vykonať meranie. Druhou možnosťou je jeho nedostupnosť včase kedy požadujeme, aby naám tieto hodnoty získal.

8.2 Určovanie polôh

V tejto sekcií sa budeme venovať určovaniu polôh uzlov siete zo získaných údajov. Vo všeobecnosti, topológia siete nebýva lokalizovaná. My však tento cieľ čiastočne máme. Z kapitoly 6.2 vieme, že presné polohy dané súradnicami vypočítať nedokážeme, pretože nemáme k dispozícií aspoň tri referenčné uzly. Na základe síl signálov môžme určiť len vzájomnú polohu uzlov, ktorú môžme prezentovať vo vlastnom súradnicovom systéme.

Pri tvorbe topológie môžeme postupovať postupne tak, že začíname od vzájomnej polohy dvoch uzlov. Následne pridávame a určujeme pozície ďalších uzlov na základe existujúcich. Hľadaný uzol by sa mal nachádzať na kružnici so stredom existujúceho, ktorý je s ním prepojený (tým bude myslené, že existuje záznam o vzájomnom dosahu). Ak vytvoríme prienik týchto kružníc, získame výslednú polohu hľadaného zariadenia. Týmto spôsobom nám vznikne systém vzájomných polôh, ktorý musí čeliť niekoľkým problémom, ako napr. problému znázornenom na obrázku 6.2. Súvisí to s tým, že odvodzovanie nasledovného uzla vždy závisí od počtu existujúcich uzlov prepojených s ním. Počty, ktoré rozlišujeme sú nasledovné:

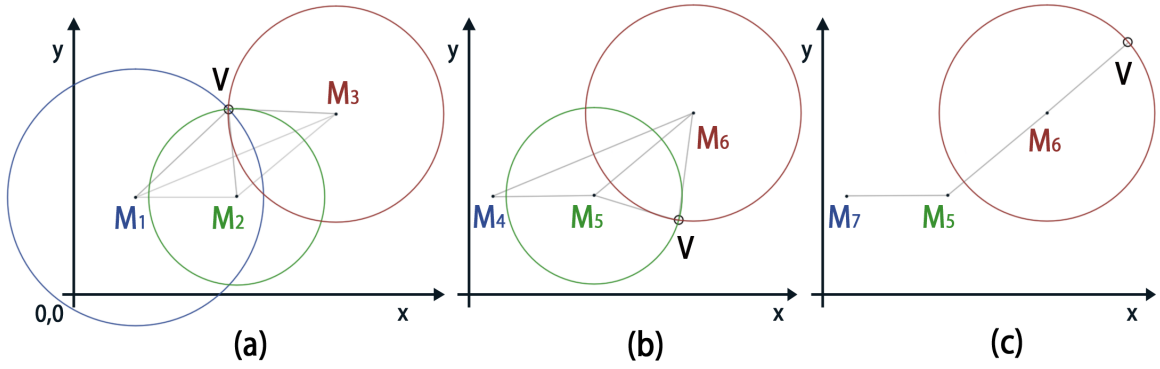
1. **Aspoň tri existujúce prepojené uzly**. V tomto prípade by mal vzniknúť jeden prienik troch kružníc *M1*, *M2*, *M3*, ktorý určí výslednú polohu *V*. Zobrazuje ho obrázok 8.1 **(a)**.

2. **Dva existujúce prepojené uzly**. Výsledkom tohto merania budú dva prieniky. Vybrať by sme mali ten, ktorý sa nachádza na vzdialenejšej pozícií od uzlov, ktoré nie sú s hľadaným uzlom prepojené. Vidíme to na obrázku 8.1 **(b)**. *M4* nemá prepojenie s hľadaným výsledkom, preto výsledok umiestnime od neho na vzdialenejšiu pozíciu.

3. **Jeden existujúci prepojený uzol**. Tento prípad je najmenej presný, pretože poloha určovaného uzlu sa môže nachádzať kdekoľvek na kružnici existujúceho. Rozumným výsledkom je opäť nájdene najvzdialenejšieho bodu kružnice od ostatných uzlov, ktoré nie sú s ním prepojené. Tento prípad znázorňuje obrázok 8.1 **(c)**. Prepojenie existuje len s *M6*, preto sa výsledok bude nachádzať v najväčšej vzdialenosti od ostatných.

Pri použití metódy postupného pridávania uzlov by sme mali postupovať v spomínanom poradí. Čím viac známych, existujúcich uzlov máme, tým bude väčšia pravdepodobnosť úspešného určenia vzájomných pozíc. Tzn. ak chceme pridať do grafu nový bod, mali by sme sa rozhodnúť pre ten, ktorý má v grafe čo najviac spojení s bodmi, na základe ktorých určíme jeho pozíciu. Tento počet existujúcich spojení môže mať viac zatiaľ neurčených uzlov zároveň. Vtedy sa opäť treba rozhodnúť pre ten, ktorý nám po jeho vložení môže opäť dopomôcť k nájdeniu ďalších uzlov. Napríklad ak sa v grafe nachádzajú tri uzly A, B, C a evidujeme tri nevložené D, E a F. Ku D a E evidujeme prepojenie so všetkými v grafe. D zároveň obsahuje prepojenie aj s F. Na základe toho sa v tomto momente rozhodneme pre pridanie uzlu D. Za ním nasleduje umiestnenie E a nakoniec F.

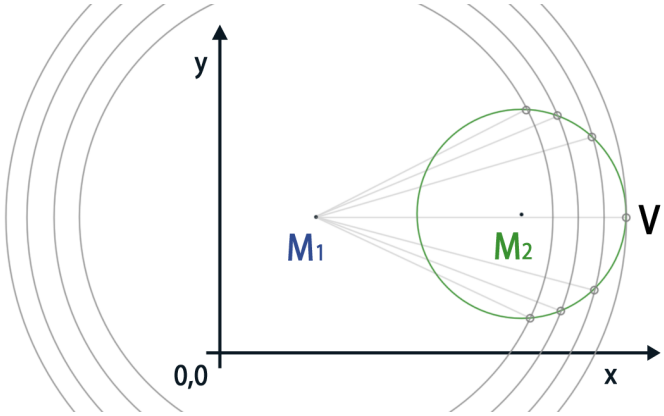
V nasledujúcich kapitolách sa budeme zaoberať riešeniami jednotlivých prípadov. Začneme z opačného konca, pretože problémy každého prípadu sa týkajú aj toho nasledovného zároveň.



Obrázek 8.1: Určenie pozície hľadaného uzla siete na základe existujúcich.

8.2.1 Riešenie pri jednom existujúcom uzle

Ak máme záznam o uzle, ktorý je prepojený len s jedným iným, riešime to väčšinou až na záver, keď už sú ostatné body určené. Jeho pozícia sa bude nachádzať na kružnici so stredom na pozícií známeho uzla a s polomerom vzdialenosti ekvivalentnej k danej sile signálu. Z toho vyplýva, že jeho pozícia je veľmi neznáma. Neostáva nám nič iné, ako umiestniť ho čo najďalej od uzlov, ktoré ho nevidia, ak vôbec nejaké sú. Obrázok 8.2 popisuje možné riešenie tohto postupu.



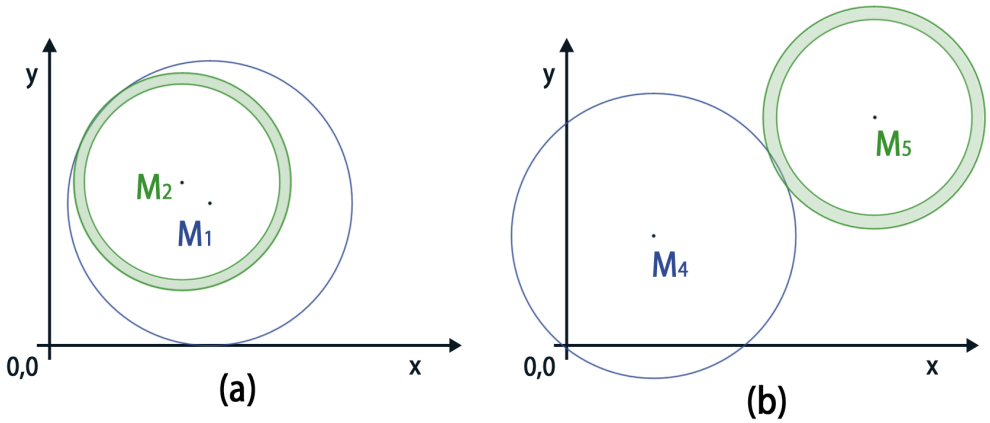
Obrázek 8.2: Riešenie nájdenia najvzdialenejšieho bodu od neprepojených uzlov.

V podstate môžme povedať, že hľadaný uzol sa bude nachádzať aj na kružnici so stredom v každom neznámom bode s neznámim polomerom. Na obrázku je to bod *M1*. Polomer zvolíme o veľkosti 0% signálu a postupne po krokoch ho zväčšujeme o zvolenú konštantu. V každom kroku nájdeme priesečníky s kružnicou známeho uzla. Priesečník, ktorý je najďalej od všetkých neznámych bude považovaný za výsledný uzol.

8.2.2 Riešenie pri dvoch existujúcich uzloch

Ak poznáme dva uzly, na základe ktorých chceme vložiť nový uzol do grafu, získame dve možnosti jeho pozície. V tomto prípade by sme mali vyhlásiť za výsledný ten, ktorého vzdialenosť je od neprepojených uzlov čo najväčšia. Tzn. stačí sumarizovať vzdialenosti od týchto dvoch bodov a rozhodnúť sa pre väčšiu z nich.

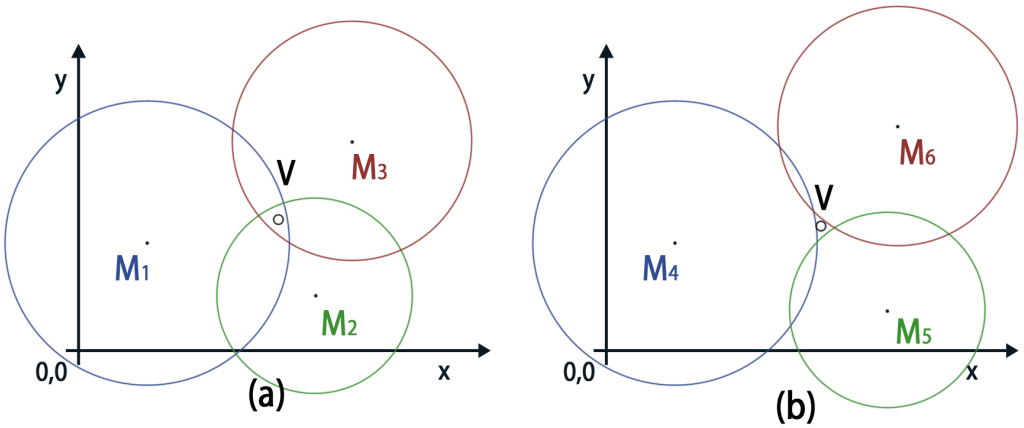
V prípade, že sa kružnice uzlov definujúce polohu hľadaného uzla nepretnú ani v jednom priesečníku, zväčšujeme polomer menšej z nich do určitého kroku tak, aby sa pretli. Ak nezískajú ani po niekoľkých krokoch prienik, považujeme to za chybu a nie je možné v určovaní topológie ďalej pokračovať. Túto situáciu popisuje 8.3. V prípade **(a)** sa kružnica *M2* nachádza vo vnútri kružnice *M1*. Po zväčšení menšej z nich získame výsledný bod, však na úkor možnej chyby. V druhom prípade **(b)** sú kružnice vedľa seba, pričom postupujeme opäť rovnako. Týmto spôsobom čiastočne riešime fyzikálne problémy, ktoré sa v prostredí vyskytujú, napr. prekážky.



Obrázek 8.3: Riešenie problémov hľadania priesečníkov dvoch kružníc pri slabom/silnom zadanom signále.

8.2.3 Riešenie pri aspoň troch existujúcich uzloch

Tento prípad je na prvý pohľad jednoduchý. Existuje niekoľko metód, ako pri ňom postupovať. Obrázok 8.4 zobrazuje dve bežné situácie, kedy sa hľadá výsledný bod pomocou vzniknutých priesečníkov. Situácie **(a)** a **(b)** obsahujú kružnice, ktoré sa pretínajú celkovo v šiestich bodoch. V prvom prípade sa výsledný bod nachádza v plochách všetkých troch kružníc, čo o druhom prípade povedať nemôžme.



Obrázek 8.4: Riešenie hľadania výsledného bodu. **(a)** príliš silný signál. **(b)** slabý signál.

Vhodný postup riešenia hľadania výsledného bodu je pomocou stredu polygónu. Jeho hraničnými bodmi budú priesečníky jednotlivých kružníc. Táto metóda dosahuje celkom priaznivé výsledky v určitých prípadoch. Však musíme riešiť situácie, kedy sa niektoré priesečníky nachádzajú veľmi ďaleko od hľadaného stredu. Tým nám vytvárajú veľkú chybu merania. Lepšie výsledky preto dosiahneme ak vytvoríme polygón len z bodov, ktoré sú od seba čo najmenej vzdialené. Stred tohto polygónu je potom pozícia uzlu, ktorého začleňujeme do grafu topológie.

Hľadanie priesečníkov troch kružníc z implementačného hľadiska riešime pomocou dvojíc kružníc. Každá dvojica väčšinou obsahuje dva priesečníky. Tzn. že sa tu nachádzajú tie isté problémy ako sú spomínané v predchádzajúcej kapitole. Riešenie pri viacerých kružniciach prebieha rovnako, pričom výsledný počet blízkych bodov bude viac, takže umiestnenie hľadaného bodu bude presnejšie.

8.3 Popis implementácie

V predchádzajúcej podkapitole sme sa dozvedeli ako získať body topológie. Okrem toho však požadujeme implementovať iné prvky, zaisťujúce správne zobrazenie celého grafu. Objekty, ktoré vykonávajú všetky tieto činnosti sú umiestnené v balíčku `util`. Môžme ich rozdeliť do troch základných kategórií:

1. Triedy na spracovanie vstupných hodnôt síl signálov. Sú to `RSSIObject`, `RSSIParser` a `RSSIRecord`.

2. JavaBean objekty popisujúce primitívne matematické útvary. Je to `GeometryObject`, od ktorej sú ďalej odvodené `PointObject`, `LineObject`, `RectangleObject` a tiež `PolygonObject`.

3. Objekty pre určenie polôh a vygenerovanie a zobrazenie výsledného grafu topológie. Sú to `TopologyMath` a `TopologyDrawer`.

Spracovanie vstupných hodnôt bolo popísané v podkapitole 8.2. Výpočetnou zložkou je tu objekt `RSSIParser`, ktorého hlavnou úlohou je vytvorenie zoznamu objektov `RSSIRecord`. Tie okrem dátumu obsahujú zoznam záznamov (typu `RSSIObject`) prepojených dvojíc doplnených o silu signálu.

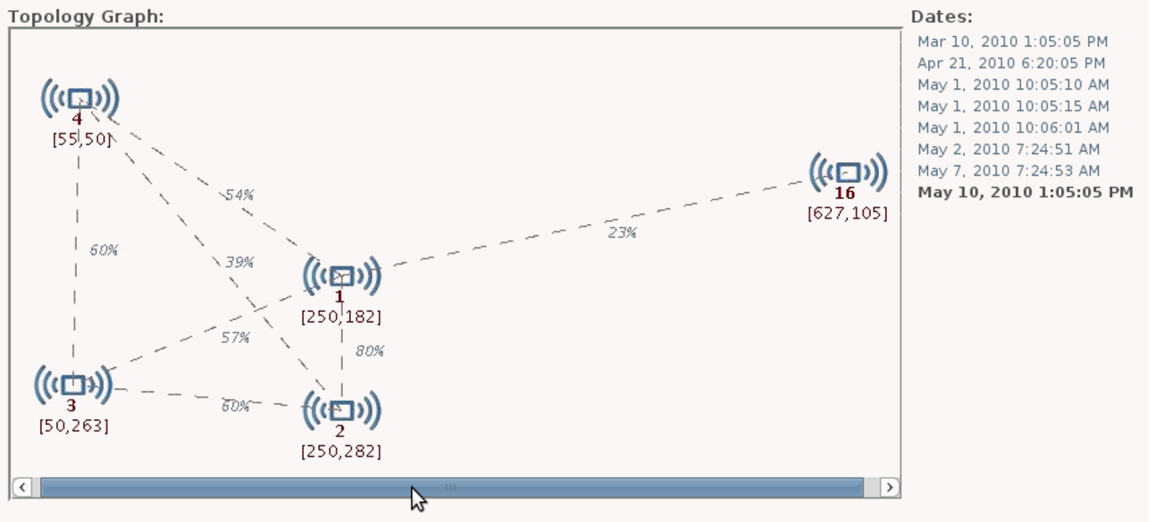
O generovanie grafu topológie sa stará trieda `TopologyMath`, ktorá pracuje s objektami primitívnych útvarov. Jej hlavnou metódou je `getTopologyPoints()`, ktorá vráti zoznam objektov typu `GeometryObject`. V ňom sa nachádzajú objekty `PointObject`, ktoré určujú výsledné body grafu topológie a objekty `LineObject`, ktoré popisujú úsečky medzi jednotlivými bodmi. Každá úsečka predstavuje vlastnosť, že dané dva body sú prepojené. Tento graf je potom pripravený na vykreslenie.

Pre účely zobrazovania bola v triede vytvorená metóda `getWindowBorder()`, ktorej výstupom je obdĺžnik s okrajmi vygenerovaného grafu. Je určená na získanie výstupného pohľadu – rozmerov a vlastností, aby sme nemuseli vykreslovať plochy, na ktorých sa prvky grafu nenachádzajú.

8.3.1 Zobrazenie topológie

Zobrazenie grafu topológie začína už zavolaním akcie `WS_Topology` odkazujúcej na triedu `WSTopology`. Tá najprv získa požadované dáta, na základe ktorých zaobstará body a úsečky grafu topológie. Jeho vykreslením sa zaoberá trieda `TopologyDrawer`, ktorá vytvorí obrázok a umiestni ho na požadované miesto disku. Tento cyklus prebieha len za podmienky, že tento graf, daný dátumom vyžiadania, ešte nebol vytvorený.

Výsledok vygenerovanej topológie zobrazuje obrázok 8.5. Môžeme tu vidieť vzájomne prepojené jednotlivé uzly siete. Každý uzol je popísaný adresou a pozíciou vo vlastnom súradnicovom systéme. Ku každej úsečke je pripísaná sila signálu medzi jej bodmi.



Obrázek 8.5: Screenshot vygenerovanej topológie.

Obrázok 8.5 tiež demonštruje výsledok určenia pozície osamoteného uzlu *16* vzhľadom na ostatné uzly *1*, *2*, *3* a *4*.

Kapitola 9

Súčasný stav a záver

Implementovali sme webové rozhranie ako informačný systém, ktorý umožňuje monitorovať rôzne udalosti v bezdrôtovej senzorovej sieti. Vytvorili sme modul grafu topológie, ktorý je súčasťou systému. V súčasnosti je systém použiteľný skôr pre vývojového pracovníka, ktorý ho môže použiť na účely pozorovania a výzkumu. Je to podmienené aj prítomnosťou jazyka ALLL, ktorý musí užívateľ poznať a použiť ho.

Výhodou tohto riešenia je použitie frameworkov, ktoré poskytujú modularitu a zapúzdrenie základných operácií. Vďaka tomu sa chyby pri implementácií vyskytovali v oveľa menšej miere, čo má vplyv aj na celkovú stabilitu. Systém bol rozdelený do vrstiev tak, aby nižšie vrstvy nevyužívali služby tých vyšších. Vďaka tomu je možné nižšie vrstvy použiť aj v iných aplikáciách, alebo vytvoriť nové vrstvy za účelom udržby softvéru.

V budúcnosti by bolo najväčším prínosom vytvorenie nových modulov. Mohli by to byť moduly na splnenie štandardných potrieb senzorovej siete, ako napr. vhodne prezentované dáta nasnímané zo senzoru, štatistické vyhodnocovanie získaných dát a pod. V konečnom dôsledku by sa malo zakryť použitie jazyka ALLL tak, aby bol softvér vhodný pre každého užívateľa.

Hlavným prínosom tejto práce bolo vytvorenie topológie siete. Skúmali sme rôzne nastavenia polohy uzlov a ich silu signálu. Výsledky však neboli veľmi presné v porovnaní so skutočným rozmiestneným uzlov, čo sa v podstate aj očakávalo. Najmä v tomto prípade by v budúcnosti bolo vhodné vykonať kalibráciu a odladenie do väčšej hĺbky a analyzovať následne dosiahnuté výsledky.

Na záver je možné skonštatovať, že vytvorený systém má veľké možnosti využitia. Bol vytvorený dobrý základ pre vybudovanie riadiaceho a kontrolného centra, ktorý by pomocou systému agentov usmerňoval a pozoroval stav rozličných zariadení, či už v priemysle alebo v každodennom živote.

Literatura

[1] Anton Belan: Analytická geometria. `http://www-kmadg.svf.stuba.sk/skripta/node18.html`, 2000.

[2] Borivoje Fuhrt, Mohammad Ilyas: *Wireless Internet Handbook: Technologies, Standards and Applications*. CRC Press, 2003, ISBN 0-8493-1502-6.

[3] CrossBow Technology, Inc: MICAz, Wireless Measurement System. `http://www.xbow.com/`.

[4] C.S. Raghavendra, Krishna M. Sivalingam, Taieb Znati: *Wireless Sensor Networks*. Springer Science+Business Media, 2004, ISBN 1-4020-7883-8.

[5] Donald Brown , Chad Michael Davis, Scott Stanlick: *Struts2 in Action*. Manning Publications, 2008, ISBN 1-933988-07-X.

[6] Feng Zhao, Leonidas Guibas: *Wireless Sensor Networks: An Information processing Approach*. Morgan Kaufmann Publishers, 2004, ISBN 1-55860-914-8.

[7] Gavin King, Christian Bauer, Max Rydahl Andersen, Emmanuel Bernard, Steve Ebersole: Hibernate Reference Documentation. `http://docs.jboss.org/hibernate/stable/core/reference/en/html/`, 2010.

[8] H. Edgar Callaway Jr.: *Wireless Sensor Networks: Architecture and Protocols*. CRC Press LLC, 2004, ISBN 0-8493-1823-8.

[9] Jan Horáček: *Platforma pro mobilní agenty v bezdrátových senzorových sítích*. Diplomová práce, FIT VUT v Brně, 2009.

[10] Jim Arlow: *UML a unifikovaný proces vývoje aplikácií*. Computer Press, Brno, 2003, ISBN 80-7226-947-X.

[11] John Rhoton: *The Wireless Internet Explained*. Digital Press, 2002, ISBN 1-55558-257-5.

[12] Karl Holger, Andreas Willig: *Protocols and Architecture for Wireless Sensor Networks*. John Wiley and Sons, 2005, ISBN 13 978-0-470-09510-2.

[13] Kazem Sohraby, Daniel Minoli, Taieb Znati: *Wireless Sensor Networks. Technology, Protocols and Applications*. Wiley-Interscience, 2007, ISBN 978-0-471-74300-2.

[14] Martin Gábor: *Informačný systém na sledovanie práce užívateľov*. Diplomová práce, FIT VUT v Brně, 2008.

[15] Mohammad Ilyas, Imad Mahgoub: *Handbook of Sensor Networks. Compact Wireless and Wired Sensing Systems.* CRC Press, 2005, ISBN 0-8493-1968-4.

[16] P. Nicopolitidis, M. S. Obaidat, G. I. Papadimitriou, A. S. Pomportsis: *Wireless Networks.* John Wiley and Sons, 2003, ISBN 0470 845295.

[17] Paul Bourke: Geometry, Surfaces, Curves, Polyhedra. `http://local.wasp.uwa.edu.au/~pbourke/geometry/`.

[18] Pavel Spáčil: *Mobilní agenti v bezdrátových senzorových sítích.* Diplomová práce, FIT VUT v Brně, 2009.

[19] Petr Hanáček: Bezdrátové a mobilní síte, podklady k prednáškam kurzu BMS. 2009.

[20] TinyOS Documentation Wiki: Mote-PC Serial communication and SerialForwarder. `http://docs.tinyos.net/`.

[21] Wolfram MathWorld: Polygon. `http://mathworld.wolfram.com/Polygon.html`.

Kapitola 10

Príloha A

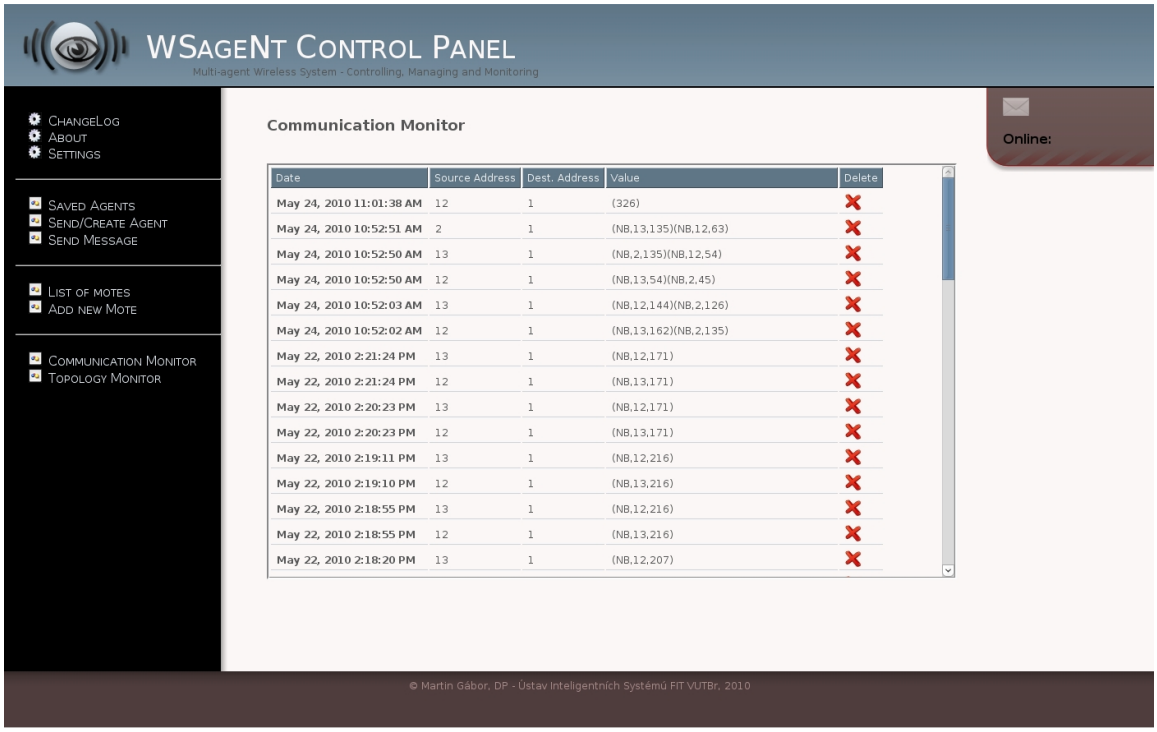
10.1 Špecifikácia prípadu užitia - Poslanie agentnej správy

ID	1
Názov	Poslanie agentnej správy
Popis	Užívateľ pošle agenta na mote so zadanou adresou.
Primárni aktéri	Užívateľ
Sekundárni aktéri	
Predpoklady	
Následné podmienky	Je zobrazená správa o výsledku poslania správy.
Akcie pre spustenie	Užívateľ vyberie akciu pre poslanie/vytvorenie agenta.
Hlavný tok	1. Systém zobrazí formulár pre vyplnenie agentnej správy. 2. Ak užívateľ vyberie jedného z existujúcich agentov 2.1. Systém vyplní hodnoty pre vybraného agenta. 3. Kým nie je vyplnená adresa zariadenia, na ktoré ma byť agent poslaný. 3.1. Užívateľ vyplní/upraví hodnoty agenta. 3.2. Potvrdí poslanie správy. 4. Systém vykoná poslanie požadovanej správy.
Alternatívne toky	Nevyplnená adresa Užvateľ zvolí uloženie agenta.
Výnimky	Neuspešné pripojenie k aplikácií, ktorá vykoná poslanie správy. Neuspešné poslanie správy. Čas určený na poslanie vypršal. Mote nie je pravdepodobne dostupný. Posielanie danému uzlu neúspešné kvôli práve prebiehajúcej komunikácií.
Frekvencia	Často
Špeciálne požiadavky	V bode 2.1. dopĺňa hodnoty vždy pri výbere iného, existujúceho agenta.

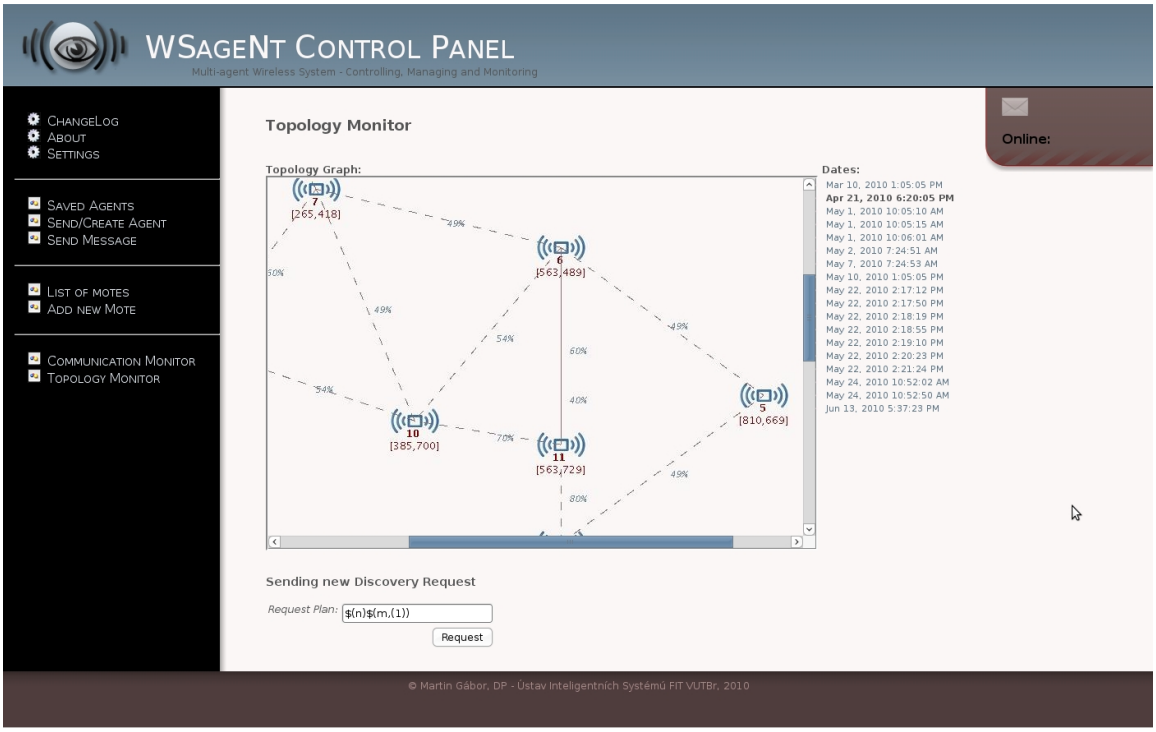
Kapitola 11

Príloha B

11.1 Screenshoty



Obrázek 11.1: Screenshot monitorovania komunikácie.



Obrázek 11.2: Screenshot tvorby topológie.

Kapitola 12

Príloha C

12.1 Obsah CD

1. Text diplomovej práce.
2. Zadanie diplomovej práce.
3. Zdrojové súbory aplikácii Control Panel a BSComm.
4. Spustiteľné súbory aplikácii Control Panel a BSComm.
5. Plagát.
6. Knižnice a inštalačný softvér.

12.2 Postup inštalácie

Tento postup je určený pre platformu Linux, na ktorej bol aj testovaný:

- Inštalácia Java Runtime Environment a Java Development Kit.
- Stiahnutie Apache Tomcat – Java kontajneru pre webové aplikácie. Nastaviť porty, na ktorých bude spustený webový lokálny server, napr. *8080*.
- Inštalácia MySQL databázového softvéru a vytvorenie databázy s názvom `wsagent`. Prístup pre užívateľa `root` s heslom `root`.
- Inštalácia softvéru TinyOs.

V systémovom prostredí sa musia nachádzať tieto premenné (treba vhodne upraviť):

```
JAVA_HOME=/usr/lib/jvm/java-6-sun
TOSDIR=/opt/tinyos-2.1.0/tos
TOSROOT=/opt/tinyos-2.1.0
MAKERULES=/opt/tinyos-2.1.0/support/make/Makerules
CLASSPATH=/opt/tinyos-2.1.0/support/sdk/java/tinyos.jar
BASEDIR=/home/tomcat
```

Pre ďalší vývoj v Netbeans IDE, importovanie nasledovných modulov a knižníc:

- Struts 2
- Hibernate
- Persistence
- mysql-connector-java-5.0.8-bin
- struts2-dojo-plugin-2.1.6.jar
- tinyos.jar

Spustenie:

- **BSComm** sa spúšťa s dvoma parametrami: cesta k sériovému portu a port k soketu.
  Príklad: `serial@/dev/ttyUSB1:iris 7777`
- **ControlPanel**:
  Vloženie `ControlPanel.war` alebo obsahu zložky `web` do kontajnera apachu Tomcat.